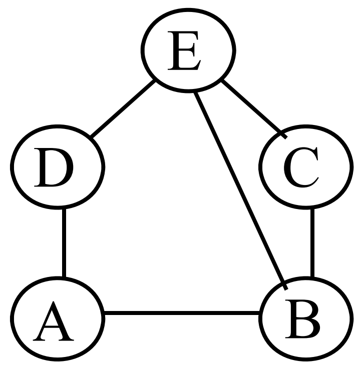
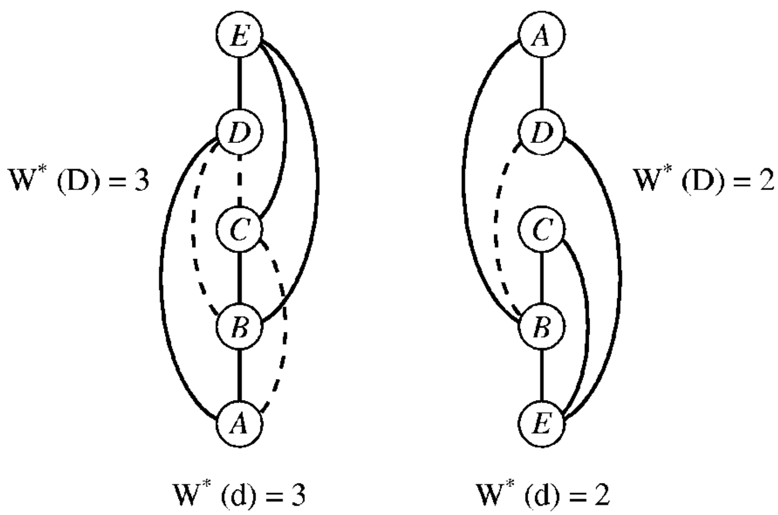
Algorithms for Reasoning with graphical models

Class4

Rina Dechter

The Induced-Width





- Width along *d*, w(d):
  - max # of previous parents
- Induced width w*(d):
  - The width in the ordered induced graph
- Induced-width w*:
  - Smallest induced-width over all orderings
- Finding w*
  - NP-complete *(Arnborg, 1985) but greedy heuristics (min-fill).*

Road Map

- Graphical models
- Constraint networks Model
- Inference
  - Variable elimination for Constraints
  - Variable elimination for CNFs
  - Greedy search for induced-width orderings
  - Variable elimination for Linear Inequalities
- Constraint propagation
- Search
- Probabilistic Networks

Finding a Small Induced-Width

- NP-complete
- A tree has induced-width of ?
- Greedy algorithms:
  - Min width
  - Min induced-width
  - Max-cardinality and chordal graphs
  - Fill-in (thought as the best)
- Anytime algorithms
  - Search-based [Gogate & Dechter 2003]
  - Stochastic (CVO) [Kask, Gelfand & Dechter 2010]

Min-width Ordering

MIN-WIDTH (MW)

input: a graph $G = (V, E)$, $V = \{v_1, ..., v_n\}$
output: A min-width ordering of the nodes $d = (v_1, ..., v_n)$.
1. **for** $j = n$ to 1 by -1 do
2. $r \leftarrow$ a node in G with smallest degree.
3. put r in position j and $G \leftarrow G - r$.
   (Delete from V node r and from E all its adjacent edges)
4. **endfor**

Proposition: *algorithm min-width finds a min-width ordering of a graph*
What is the Complexity of MW?
O(e)

Greedy Orderings Heuristics

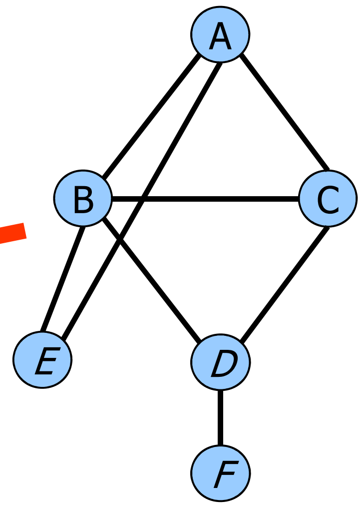
- Min-induced-width
  - From last to first, pick a node with smallest width, then connect parent and remove

- Min-Fill
  - From last to first, pick a node with smallest fill-edges

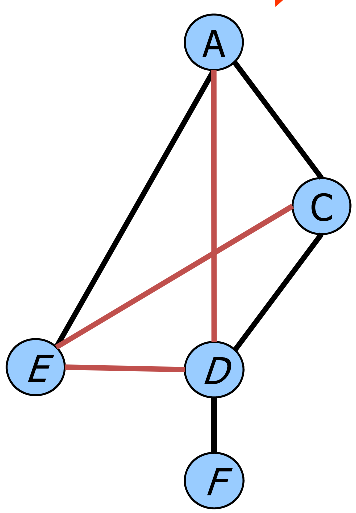
Complexity? $O(n^3)$

Min-Fill Heuristic

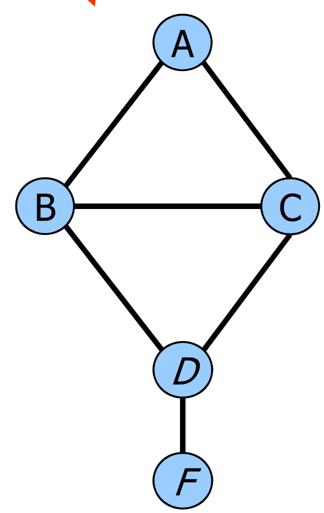
- Select the variable that creates the fewest "fill-in" edges



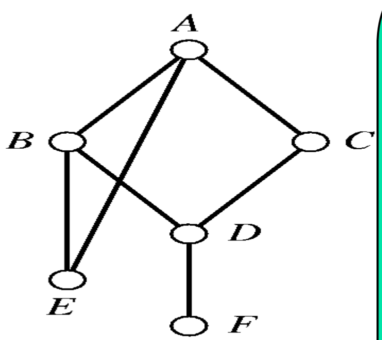
Eliminate B next?
Connect neighbors
"Fill-in" = 3:
(A,D), (C,E), (D,E)



Eliminate E next?
Neighbors already connected
"Fill-in" = 0

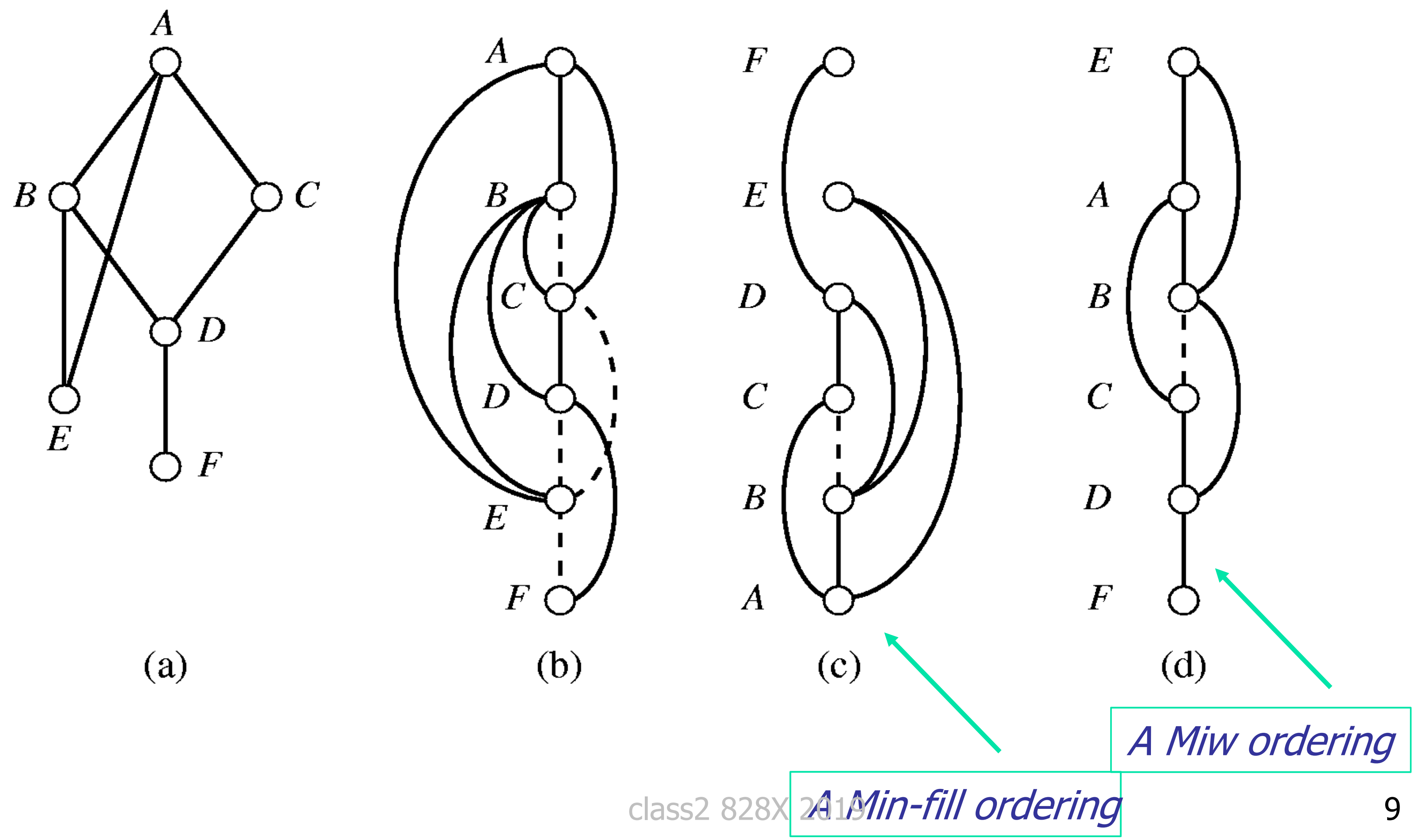


Example



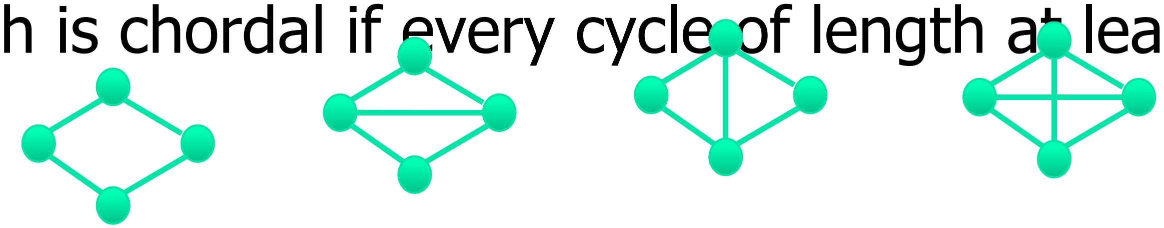
(a)

Different Induced-Graphs



Chordal Graphs

- A graph is chordal if every cycle of length at least 4 has a chord



- Deciding chordality by max-cardinality ordering:
  - from 1 to n, always assigning a next node connected to a largest set of previously selected nodes.

- A graph along max-cardinality order has no fill-in edges iff it is chordal.

- The maximal cliques of chordal graphs form a tree

[Tarjan & Yanakakis 1980]

Greedy Orderings Heuristics

- Min-Induced-width
  - From last to first, pick a node with smallest width

- Min-Fill
  - From last to first, pick a node with smallest fill-edges

    Complexity? $O(n^3)$

- Max-Cardinality search *[Tarjan & Yanakakis 1980]*
  - From **first to last**, pick a node with largest neighbors already ordered. *Complexity?* $O(n+m)$

Max-cardinality ordering

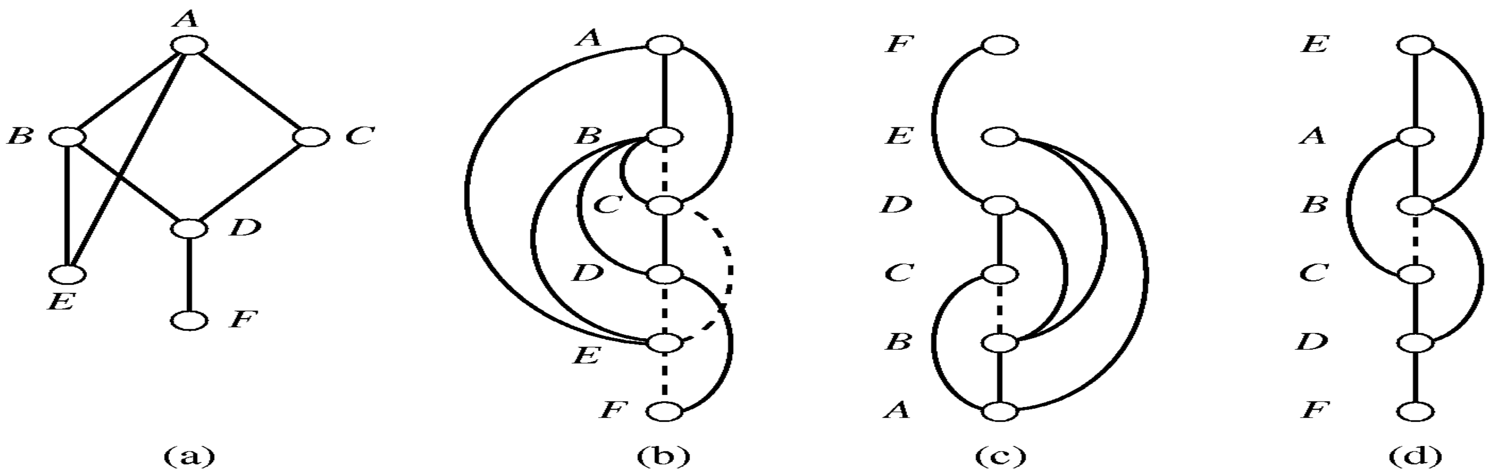
MAX-CARDINALITY (MC)

input: a graph $G = (V, E)$, $V = \{v_1, ..., v_n\}$
output: An ordering of the nodes $d = (v_1, ..., v_n)$.
1. Place an arbitrary node in position 0.
2. **for** $j = 1$ to n do
3. $r \leftarrow$ a node in G that is connected to a largest subset of nodes in positions 1 to $j - 1$, breaking ties arbitrarily.
4. **endfor**

Proposition 5.3.3 *[56] Given a graph $G = (V, E)$ the complexity of max-cardinality search is $O(n + m)$ when $|V| = n$ and $|E| = m$.*

Example

We see again that *G* in the Figure (a) is not chordal since the parents of *A* are not connected in the max-cardinality ordering in Figure (d). If we connect *B* and *C*, the resulting induced graph is chordal.



Which Greedy Algorithm is Best?

- Min-Fill, prefers a node who add the least number of fill-in arcs.

- Empirically, fill-in is the best among the greedy algorithms (MW,MIW,MF,MC)

- Complexity of greedy orderings?
- MW is O(e), MIW: O(n^3) MF O(n^3) MC is O(e+n)

K-trees

Definition 5.3.4 (k-trees) *A subclass of chordal graphs are k-trees. A k-tree is a chordal graph whose maximal cliques are of size $k+1$, and it can be defined recursively as follows: (1) A complete graph with k vertices is a k-tree. (2) A k-tree with r vertices can be extended to $r+1$ vertices by connecting the new vertex to all the vertices in any clique of size k. A partial k-tree is a k-tree having some of its arcs removed. Namely it will clique of size smaller than k.*

Finding a Small Induced-Width

- NP-complete
- A tree has induced-width of ?
- Greedy algorithms:
  - Min width (MW)
  - Min induced-width (MIW)
  - Max-cardinality and chordal graphs (MC)
  - Min-Fill (thought as the best) (MIN-FILL)
- Anytime algorithms
  - Search-based [Gogate & Dechter 2003]
  - Stochastic (CVO) [Kask, Gelfand & Dechter 2010]

Summary Of Inference Scheme

- Bucket elimination is time and memory exponential in the induced-width.
- Finding the w* is hard, but greedy schemes work quit well to approximate. Most popular is fill-edges
- W(d) is the induced-width along an ordering d. Smallest induced-width is also called tree-width.

Recent work in my group

- **Vibhav Gogate and Rina Dechter.** "A Complete Anytime Algorithm for Treewidth". *In UAI 2004.*
- **Andrew E. Gelfand, Kalev Kask, and Rina Dechter.** "Stopping Rules for Randomized Greedy Triangulation Schemes" in *Proceedings of AAAI 2011.*
- Kask, Gelfand and Dechter, BEEM: Bucket Elimination with External memory, AAAI 2011 or UAI 2011

- Potential project

Greedy Algorithms for Induced-Width

- Min-width ordering
- Min-induced-width ordering
- Max-cardinality ordering
- Min-fill ordering
- Chordal graphs
- Hypergraph partitionings

(Project: present papers on induced-width, run algorithms for induced-width on new benchmarks…)

Min-width Ordering

MIN-WIDTH (MW)

input: a graph $G = (V, E)$, $V = \{v_1, ..., v_n\}$
output: A min-width ordering of the nodes $d = (v_1, ..., v_n)$.
1. **for** $j = n$ to 1 by -1 do
2. $\quad r \leftarrow$ a node in G with smallest degree.
3. $\quad$ put r in position j and $G \leftarrow G - r$.
   (Delete from V node r and from E all its adjacent edges)
4. **endfor**

Proposition: *algorithm min-width finds a min-width ordering of a graph*
Complexity:?
O(e)

Greedy Orderings Heuristics

min-induced-width (miw)
input: a graph G = (V;E), V = {v1; :::; vn}
output: A miw ordering of the nodes d = (v1; :::; vn).
1. for j = n to 1 by -1 do
2. r ← a node in V with smallest degree.
3. put r in position j.
4. connect r's neighbors: E ← E union {(vi; vj)| (vi; r) in E; (vj ; r) 2 in E},
5. remove r from the resulting graph: V ←V - {r}.

Theorem: *A graph is a tree iff it has both width and induced-width of 1.*

min-fill (min-fill)
input: a graph G = (V;E), V = {v1; :::; vn}
output: An ordering of the nodes d = (v1; :::; vn).
1. for j = n to 1 by -1 do
2. r ←a node in V with smallest fill edges for his parents.
3. put r in position j.
4. connect r's neighbors: E ←E union {(vi; vj)| (vi; r) 2 E; (vj ; r) in E},
5. remove r from the resulting graph: V ←V –{r}.

Chordal Graphs; Max-Cardinality Ordering

- A graph is chordal if every cycle of length at least 4 has a chord
- Finding w* over chordal graph is easy using the max-cardinality ordering.
- The induced graph is chordal
- K-trees are special chordal graphs.
- Finding the max-clique in chordal graphs is easy (just enumerate all cliques in a max-cardinality ordering

Road Map

- Graphical models
- Constraint networks Model
- Inference
  - Variable elimination for Constraints
  - Variable elimination for CNFs
  - Greedy search for induced-width orderings
  - Variable elimination for Linear Inequalities
- Constraint propagation
- Search
- Probabilistic Networks

Linear Inequalities

Variables domains are the real numbers

$$(3x_i + 2x_j \leq 3) \wedge (-4x_i + 5x_j \leq 1)$$

Definition 3.3.1 (Linear elimination) *Let* $\alpha = \sum_{i=1}^{(r-1)} a_i x_i + a_r x_r \leq c$, *and* $\beta = \sum_{i=1}^{(r-1)} b_i x_i + b_r x_r \leq d$. *Then* $elim_r(\alpha, \beta)$ *is applicable only if* a_r *and* b_r *have opposite signs, in which case* $elim_r(\alpha, \beta) = \sum_{i=1}^{r-1}(-a_i \frac{b_r}{a_r} + b_i)x_i \leq -\frac{b_r}{a_r}c + d$. *If* a_r *and* b_r *have the same sign the elimination implicitly generates the universal constraint.*

Linear Inequalities: Fourier Elimination

DIRECTIONAL-LINEAR-ELIMINATION (φ, d)

Input: A *set of linear inequalities* φ, an ordering $d = x_1, \ldots, x_n$.

Output: A decision of whether φ is satisfiable. If it is, a backtrack-free theory $E_d(\varphi)$.

1. **Initialize:** Partition inequalities into ordered buckets.
2. **for** $i \leftarrow n$ **downto** 1 **do**
3. **if** x_i has one value in its domain **then**
   . substitute the value into each inequality in the bucket and put the resulting inequality in the right bucket.
4. **else,for each** pair $\{\alpha, \beta\} \subseteq bucket_i$, compute $\gamma = elim_i(\alpha, \beta)$
   if γ has no solutions, return $E_d(\varphi) = \{\}$, "inconsistency"
   else add γ to the appropriate lower bucket.
5. **return** $E_d(\varphi) \leftarrow \bigcup_i bucket_i$

Directional linear elimination, DLE : generates a backtrack-free representation

Theorem 4.8.3 *Given a set of linear inequalities φ, algorithm DLE (Fourier elimination) decides the consistency of φ over the Rationals and the Reals, and it generates an equivalent backtrack-free representation.* $\Box$

Example

$$\begin{aligned}
bucket_4 &: \quad 5x_4 + 3x_2 - x_1 \leq 5,\ x_4 + x_1 \leq 2,\ -x_4 \leq 0, \\
bucket_3 &: \quad x_3 \leq 5,\ x_1 + x_2 - x_3 \leq -10 \\
bucket_2 &: \quad x_1 + 2x_2 \leq 0. \\
bucket_1 &:
\end{aligned}$$

Figure 4.23: initial buckets

Example

$$\begin{aligned} bucket_4 &: \quad 5x_4 + 3x_2 - x_1 \le 5, \;\; x_4 + x_1 \le 2, \;\; -x_4 \le 0, \\ bucket_3 &: \quad x_3 \le 5, \;\; x_1 + x_2 - x_3 \le -10 \\ bucket_2 &: \quad x_1 + 2x_2 \le 0. \\ bucket_1 &: \end{aligned}$$

Figure 4.23: initial buckets

$$\begin{aligned} bucket_4 &: \quad 5x_4 + 3x_2 - x_1 \le 5, \;\; x_4 + x_1 \le 2, \;\; -x_4 \le 0, \\ bucket_3 &: \quad x_3 \le 5, \;\; x_1 + x_2 - x_3 \le -10 \\ bucket_2 &: \quad x_1 + 2x_2 \le 0 \;||\; 3x_2 - x_1 \le 5, x_1 + x_2 \le -5 \\ bucket_1 &: \quad ||\; x_1 \le 2. \end{aligned}$$

Figure 4.24: final buckets

Algorithms for Reasoning with graphical models

Class5

Rina Dechter

Road Map

- Graphical models
- Constraint networks Model
- Inference
  - Variable elimination for Constraints
  - Variable elimination for CNFs
  - Variable elimination for Linear Inequalities
  - Constraint propagation (chapter 3 Dechter2)
- Search
- Probabilistic Networks

Outline

- Arc-consistency algorithms
- Path-consistency and i-consistency
- Arc-consistency, Generalized arc-consistency, relational arc-consistency
- Global and bound consistency
- Distributed (generalized) arc-consistency
- Consistency operators: join, resolution, Gausian elimination

Sudoku – Approximation: Constraint Propagation

- *Constraint*
- *Propagation*

- *Inference*

		2	4		6			
8	6	5	1			2		
	1				8	6		9
9				4		8	6	
	4	7				1	9	
	5	8		6				3
4		6	9				7	~~2,3,4,6~~ 2
		9			4	5	8	1
			3		2	9		

- ***Variables**: empty slots*
- ***Domains** = {1,2,3,4,5,6,7,8,9}*
- ***Constraints**:*
  - *27 all-different*

Each row, column and major block must be alldifferent

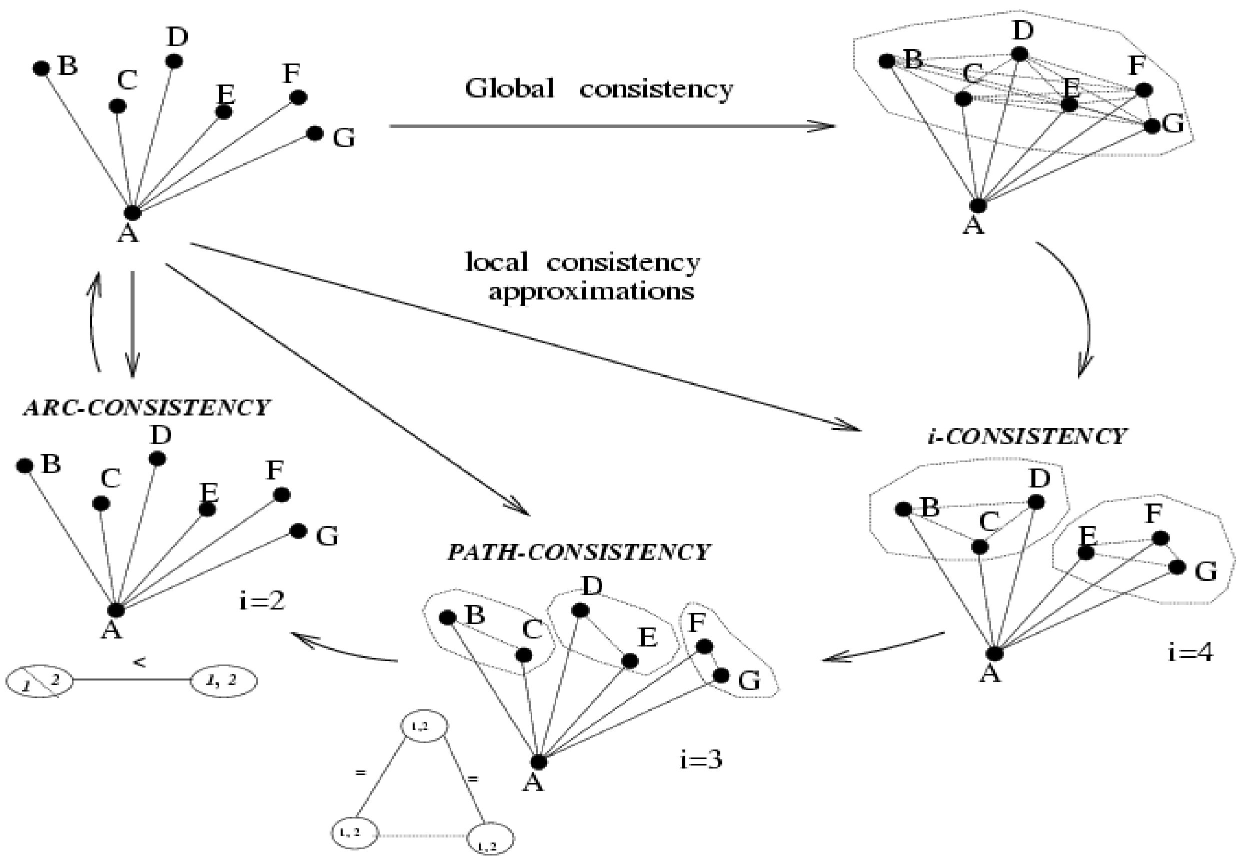
"Well posed" if it has unique solution: 27 constraints

Approximating Inference: Local Constraint Propagation

- Problem: Adaptive-consistency/Bucket-elimination algorithms are intractable when *induced-width* is large

- Approximation: bound the size of recorded dependencies, i.e. perform local constraint propagation (local inference)

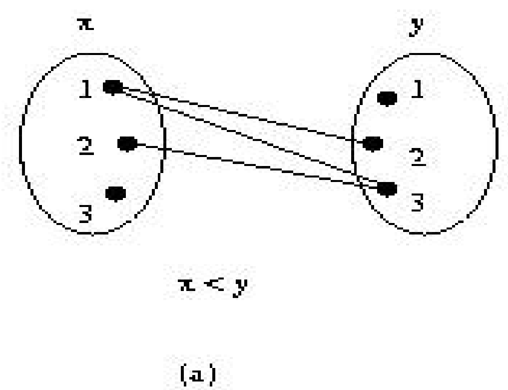
From Global to Local Consistency

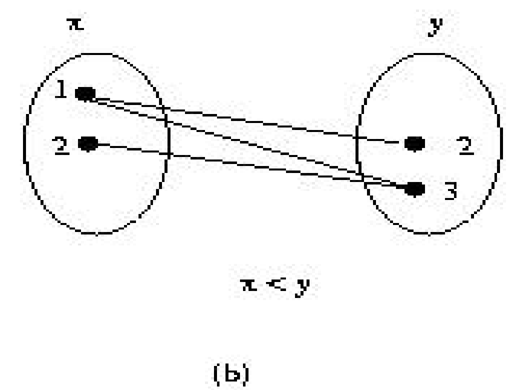


Arc-Consistency

A binary constraint R(X,Y) is arc-consistent w.r.t. X is every value In X's domain has a match in Y's domain.

$$R_X = \{1,2,3\},\ R_Y = \{1,2,3\},\ \text{constraint}\quad X < Y$$





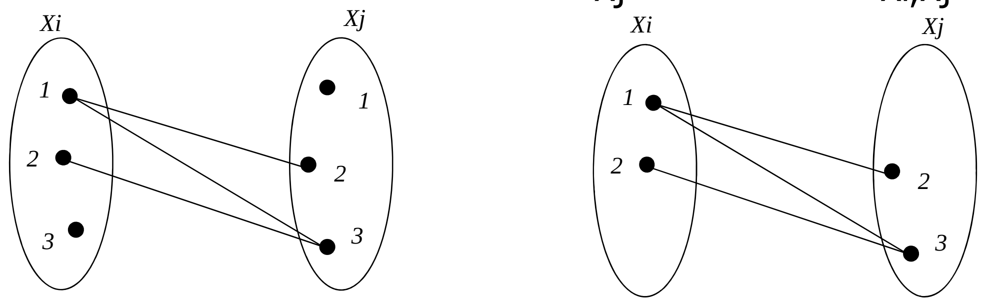
Only domains are reduced:

$$R_X \leftarrow \prod_X R_{XY} \bowtie D_Y$$

Arc-Consistency

Definition: Given a constraint graph ***G***,

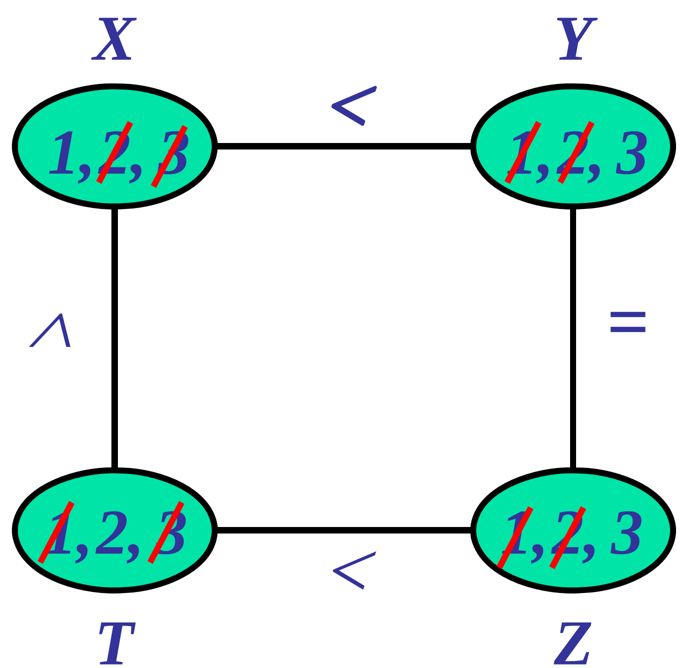
- A variable X_i is arc-consistent relative to X_j iff for every value $a \in D_{Xi}$ there exists a value $b \in D_{Xj}$ | $(a, b) \in R_{Xi,Xj}$.



- The constraint $R_{Xi,Xj}$ is arc-consistent iff
  - X_i is arc-consistent relative to X_j and
  - X_j is arc-consistent relative to X_i.
- A binary CSP is arc-consistent iff every constraint (or sub-graph of size 2) is arc-consistent.

Arc-consistency

$$1 \le X, Y, Z, T \le 3$$
$$X < Y$$
$$Y = Z$$
$$T < Z$$
$$X \le T$$



Question: What will be the domain of Y once the network is arc-consistent? Or, how many values will it have?

Arc-consistency

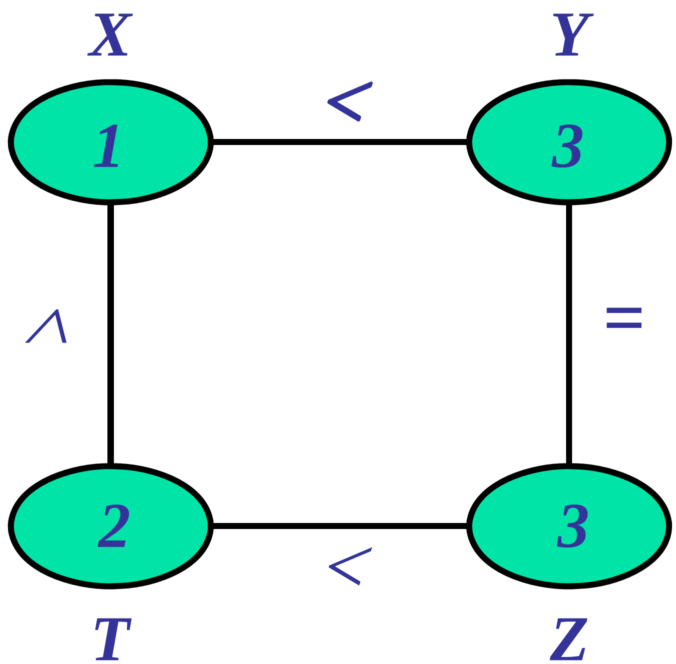
$1 \leq X, Y, Z, T \leq 3$

$X < Y$

$Y = Z$

$T < Z$

$X \leq T$



$$R_X \leftarrow \prod_X R_{XY} \bowtie D_Y$$

Revise for Arc-Consistency

REVISE$((x_i), x_j)$

input: a subnetwork defined by two variables $X = \{x_i, x_j\}$, a distinguished variable x_i, domains: D_i and D_j, and constraint R_{ij}

output: D_i, such that, x_i arc-consistent relative to x_j

1. **for** each $a_i \in D_i$
2. **if** there is no $a_j \in D_j$ such that $(a_i, a_j) \in R_{ij}$
3. **then** delete a_i from D_i
4. **endif**
5. **endfor**

$\bowtie = \otimes$

Figure 3.2: The Revise procedure

$$D_i \leftarrow D_i \cap \pi_i (R_{ij} \otimes D_j)$$

Revise for Arc-Consistency

REVISE$((x_i), x_j)$

input: a subnetwork defined by two variables $X = \{x_i, x_j\}$, a distinguished variable x_i, domains: D_i and D_j, and constraint R_{ij}

output: D_i, such that, x_i arc-consistent relative to x_j

1. **for** each $a_i \in D_i$
2. **if** there is no $a_j \in D_j$ such that $(a_i, a_j) \in R_{ij}$
3. **then** delete a_i from D_i
4. **endif**
5. **endfor**

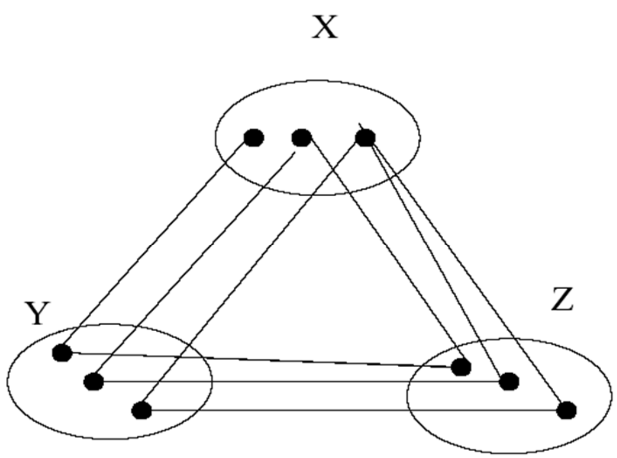
Complexity?

Figure 3.2: The Revise procedure

$O(k^2)$

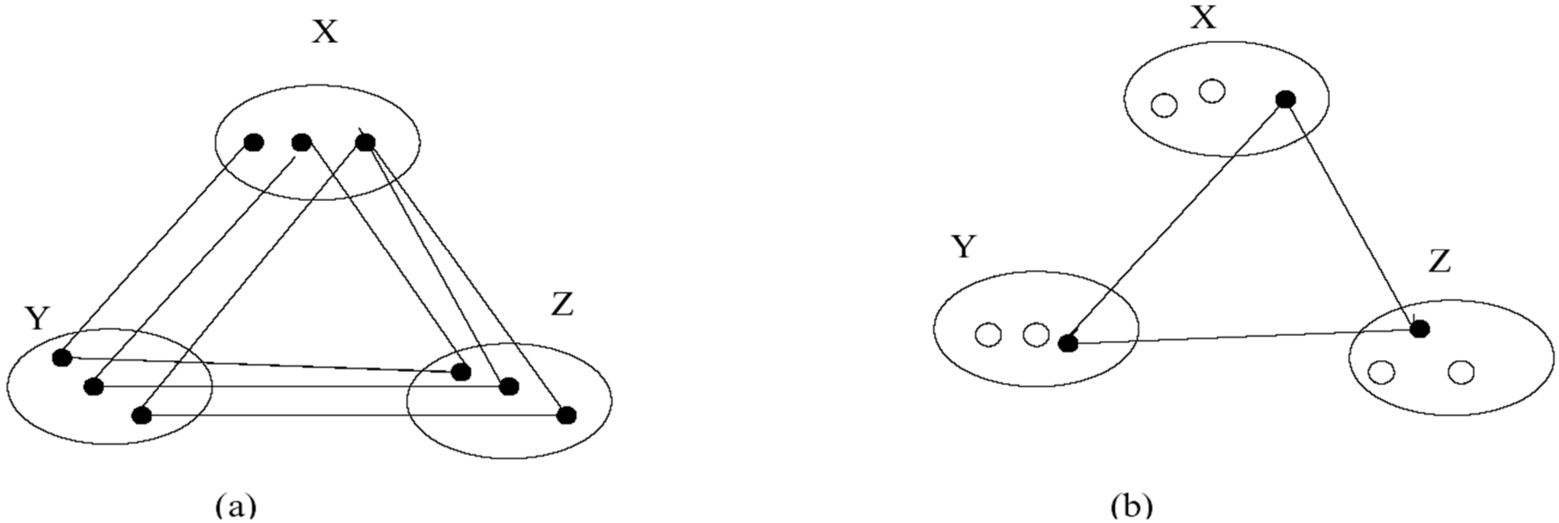
$$D_i \leftarrow D_i \cap \pi_i (R_{ij} \bowtie D_j)$$

A matching diagram describing a network of constraints that is not arc-consistent (b) An arc-consistent equivalent network.



(a)

A matching diagram describing a network of constraints that is not arc-consistent (b) An arc-consistent equivalent network.



AC-1

AC-1($\mathcal{R}$)

input: a network of constraints $\mathcal{R} = (X, D, C)$
output: $\mathcal{R}'$ which is the loosest arc-consistent network equivalent to $\mathcal{R}$

1. **repeat**
2.   **for** every pair $\{x_i, x_j\}$ that participates in a constraint
3.   Revise($(x_i), x_j$) (or $D_i \leftarrow D_i \cap \pi_i(R_{ij} \bowtie D_j)$)
4.   Revise($(x_j), x_i$) (or $D_j \leftarrow D_j \cap \pi_j(R_{ij} \bowtie D_i)$)
5.   **endfor**
6. **until** no domain is changed

Figure 3.4: Arc-consistency-1 (AC-1)

- Proof:
  - Convergence?
  - Completeness?

AC-1

AC-1($\mathcal{R}$)

input: a network of constraints $\mathcal{R} = (X, D, C)$
output: $\mathcal{R}'$ which is the loosest arc-consistent network equivalent to $\mathcal{R}$

1. **repeat**
2. **for** every pair $\{x_i, x_j\}$ that participates in a constraint
3. Revise$((x_i), x_j)$ (or $D_i \leftarrow D_i \cap \pi_i(R_{ij} \bowtie D_j)$)
4. Revise$((x_j), x_i)$ (or $D_j \leftarrow D_j \cap \pi_j(R_{ij} \bowtie D_i)$)
5. **endfor**
6. **until** no domain is changed

Figure 3.4: Arc-consistency-1 (AC-1)

- Complexity (Mackworth and Freuder, 1986):
- *e* = number of arcs, *n* variables, *k* values
- (ek^2 each loop, *nk* number of loops), best-case = *ek*
- Arc-consistency is: $\Omega(ek^2)$
- Complexity of AC-1: O(enk^3)

AC-3

AC-3($\mathcal{R}$)

input: a network of constraints $\mathcal{R} = (X, D, C)$
output: $\mathcal{R}'$ which is the largest arc-consistent network equivalent to $\mathcal{R}$

for every pair $\{x_i, x_j\}$ that participates in a constraint $R_{ij} \in \mathcal{R}$
    $queue \leftarrow queue \cup \{(x_i, x_j), (x_j, x_i)\}$
endfor
while $queue \neq \{\}$
    select and delete (x_i, x_j) from *queue*
    $Revise((x_i), x_j)$
    if $Revise((x_i), x_j)$ causes a change in D_i
        then $queue \leftarrow queue \cup \{(x_k, x_i), i \neq k\}$
    endif
endwhile

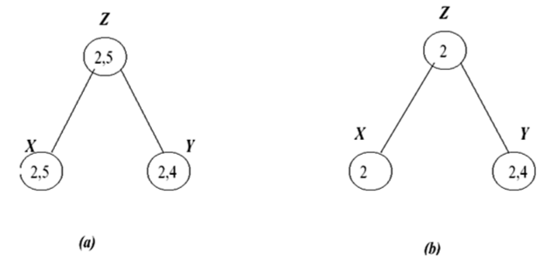
Figure 3.5: Arc-consistency-3 (AC-3)

- Complexity: $O(ek^3)$
- Best case O(ek), since each arc may be processed in O(2k)

Exercise: Apply Arc-Consistency in Class

- Draw the network's primal and dual constraint graph
- Network =
  - Domains {1,2,3,4}
  - Constraints: $y < x$, $z < y$, $t < z$, $f<t$, $x<=t+1$, $Y<f+2$
  - Apply AC-3?

AC-4 (just FYI)




```
AC-4(R)
input: a network of constraints R
output: An arc-consistent network equivalent to R
Initialization: M ← ∅,
    initialize S(xi,ci), counter(i, ai, j) for all Rij
    for all counters
        if counter(xi, ai, xj) = 0 (if < xi, ai > is unsupported by xj)
            then add < xi, ai > to LIST
        endif
    endfor
while LIST is not empty
    choose < xi, ai > from LIST, remove it, and add it to M
    for each < xj, aj > in S(xi,ai)
        decrement counter(xj, aj, xi)
        if counter(xj, aj, xi) = 0
            then add < xj, aj > to LIST
        endif
    endfor
endwhile
```

Figure 3.7: Arc-consistency-4 (AC-4)

- Complexity: $O(ek^2)$
- (Counter is the number of supports to a_i in x_i from x_j. $S_{(xi,ai)}$ is the set of pairs that (x_i, a_i) supports)

Example applying AC-4

Example 3.2.9 Consider the problem in Figure 3.6. Initializing the $S_{(x,a)}$ arrays (indicating all the variable-value pairs that each $< x, a >$ supports), we have :
$S_{(z,2)} = \{< x, 2 >, < y, 2 >, < y, 4 >\}$, $S_{(z,5)} = \{< x, 5 >\}$, $S_{(x,2)} = \{< z, 2 >\}$, $S_{(x,5)} = \{< z, 5 >\}$, $S_{(y,2)} = \{< z, 2 >\}$, $S_{(y,4)} = \{< z, 2 >\}$.
For counters we have: $counter(x, 2, z) = 1$, $counter(x, 5, z) = 1$, $counter(z, 2, x) = 1$, $counter(z, 5, x) = 1$, $counter(z, 2, y) = 2$, $counter(z, 5, y) = 0$, $counter(y, 2, z) = 1$, $counter(y, 4, z) = 1$. (Note that we do not need to add counters between variables that are not directly constrained, such as x and y.) Finally, $List = \{< z, 5 >\}$, $M = \emptyset$. Once $< z, 5 >$ is removed from $List$ and placed in M, the counter of $< x, 5 >$ is updated to $counter(x, 5, z) = 0$, and $< x, 5 >$ is placed in $List$. Then, $< x, 5 >$ is removed from $List$ and placed in M. Since the only value it supports is $< z, 5 >$ and since $< z, 5 >$ is already in M, the $List$ remains empty and the process stops. □

Arc-Consistency Algorithms

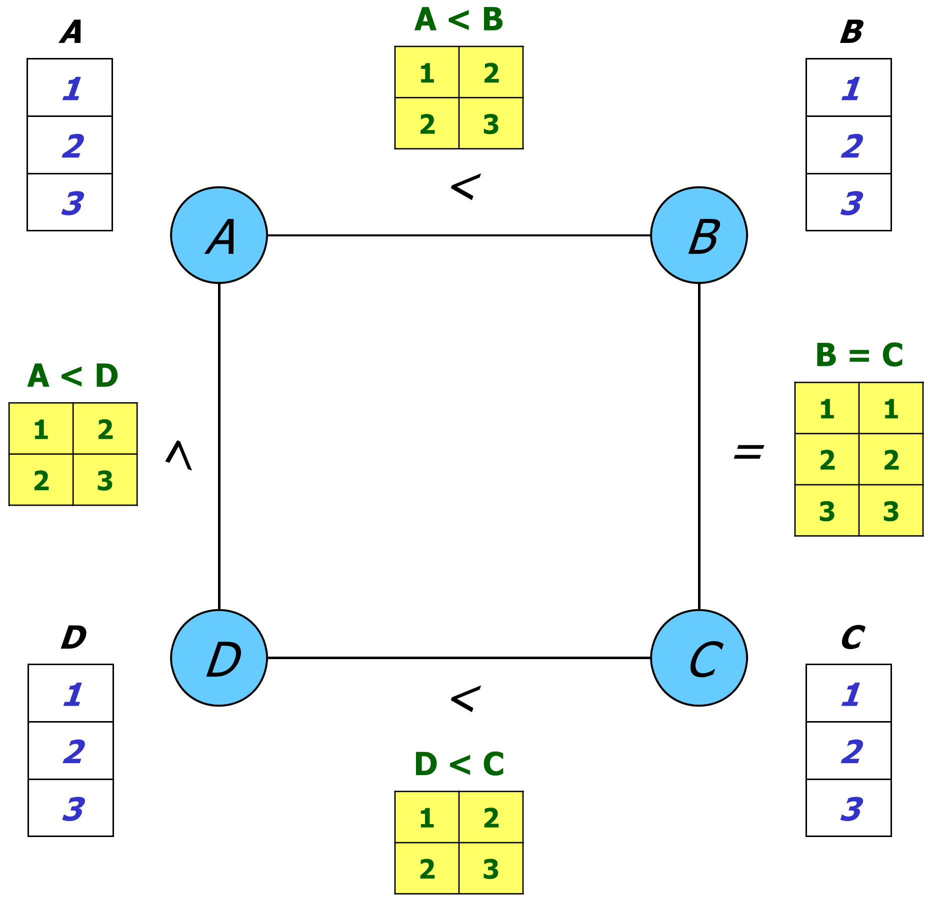
- AC-1: brute-force, distributed $O(nek^3)$
- AC-3, queue-based $O(ek^3)$
- AC-4, context-based, optimal $O(ek^2)$
- AC-5,6,7,…. Good in special cases
- Important: applied at every node of search

n=number of variables, e=#constraints, k=domain size

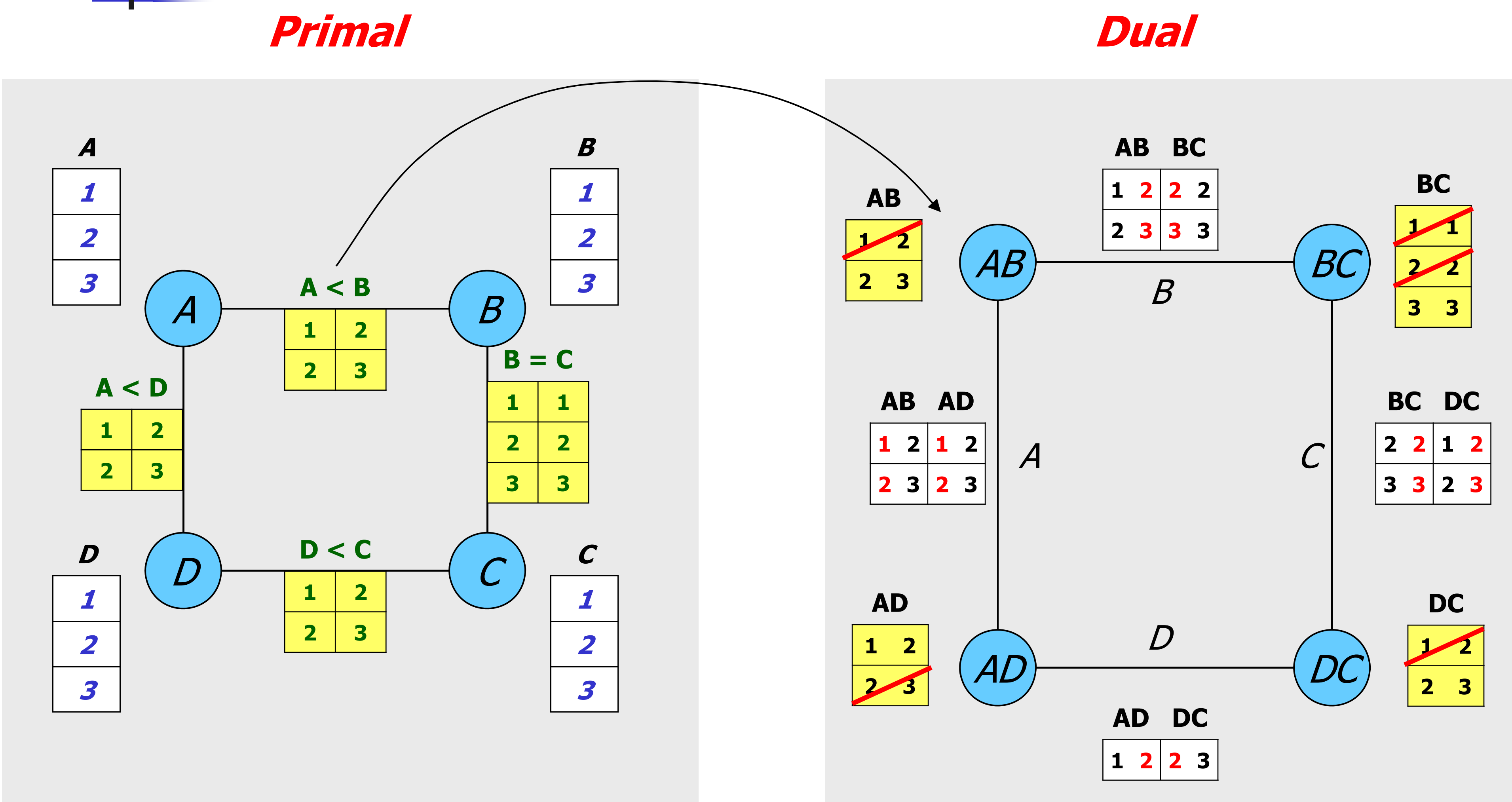
Mackworth and Freuder (1977,1983), Mohr and Anderson, (1985)…

From Arc-Consistency to Relational Arc-Consistency

- Sound
- Incomplete
- Always converges (polynomial)



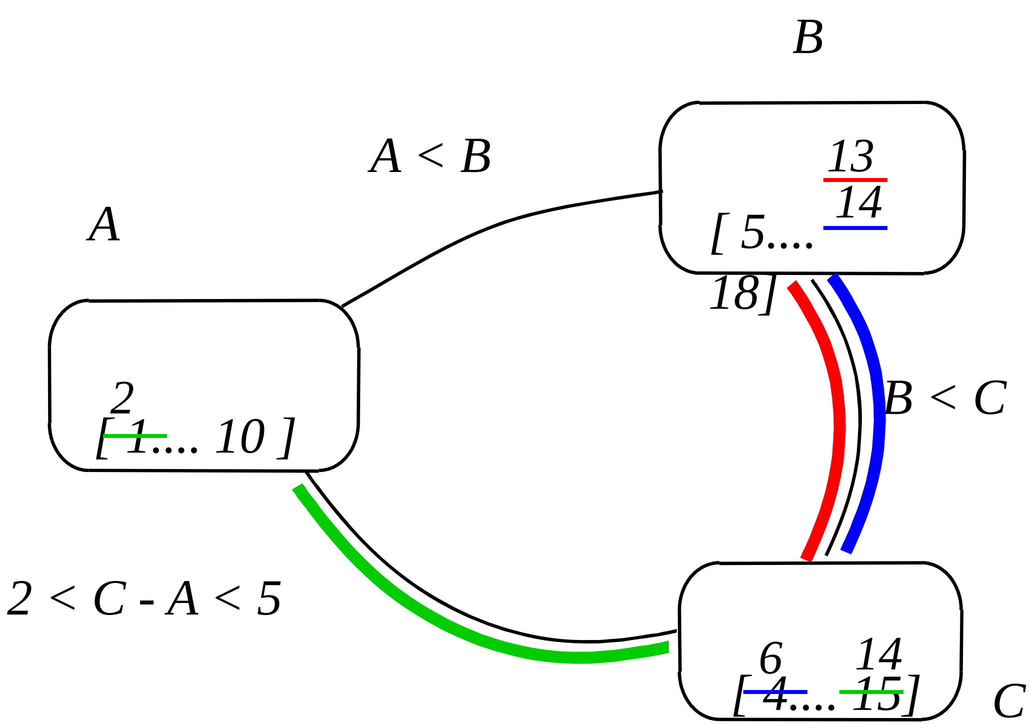
Relational Distributed Arc-Consistency



All Arc-consistent algorithms converge to an equivalent and loosest arc-consistent network!!!

Constraint Checking

- Arc-consistency



1- B: [5 .. 14]

C: [6 .. 15]

2- A: [2 .. 10]

C: [6 .. 14]

3- B: [5 .. 13]

Is Arc-Consistency Enough?

- Example: a triangle graph-coloring with 2 values.
  - Is it arc-consistent?
  - Is it consistent?
- It is not path, or 3-consistent.

Outline

- Arc-consistency algorithms
- Path-consistency and i-consistency
- Arc-consistency, Generalized arc-consistency, relation arc-consistency
- Global and bound consistency
- Distributed (generalized) arc-consistency
- Consistency operators: join, resolution, Gausian elimination

Path-Consistency

- A pair (x, y) is path-consistent relative to Z, if every consistent assignment (x, y) has a consistent extension to z.

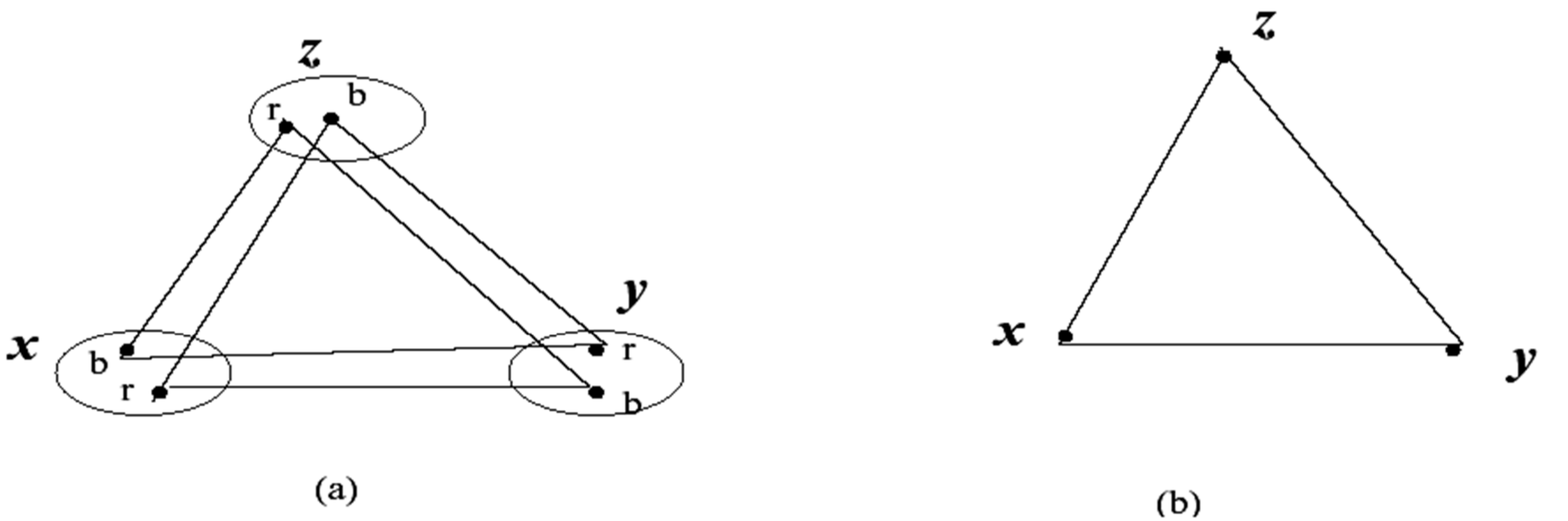


Figure 3.8: (a) The matching diagram of a 2-value graph coloring problem. (b) Graphical picture of path-consistency using the matching diagram.

Example: Path-Consistency

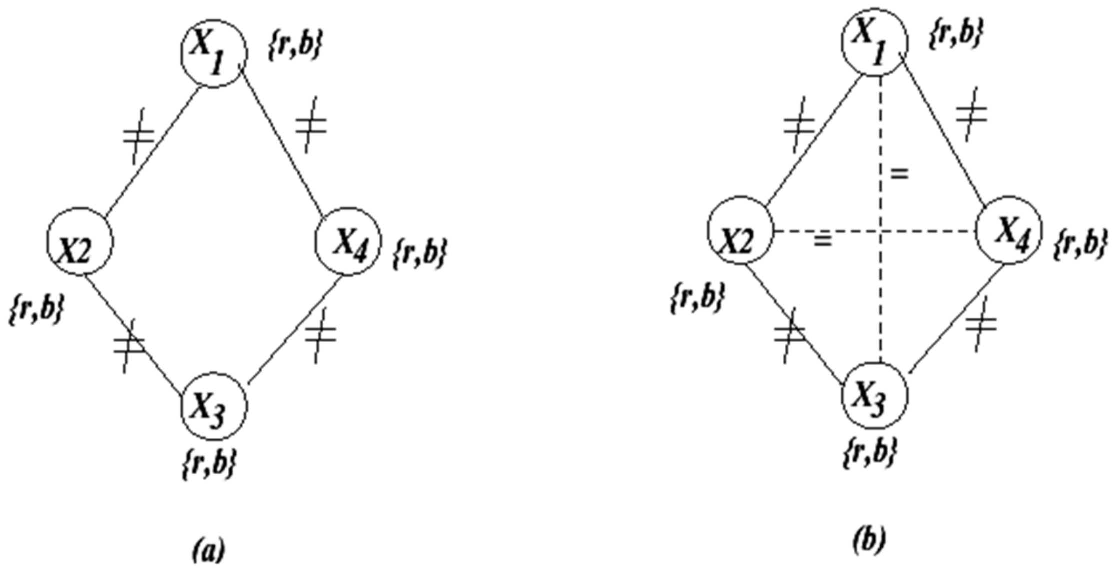


Figure 3.12: A graph-coloring graph (a) before path-consistency (b) after path-consistency

Revise-3

$\text{REVISE-3}((x,y),z)$

input: a three-variable subnetwork over (x,y,z), R_{xy}, R_{yz}, R_{xz}.
output: revised R_{xy} path-consistent with z.

1. **for** each pair $(a,b) \in R_{xy}$
2.   **if** no value $c \in D_z$ exists such that $(a,c) \in R_{xz}$ and $(b,c) \in R_{yz}$
3.   **then** delete (a,b) from R_{xy}.
4.   **endif**
5. **endfor**

Figure 3.9: Revise-3

$$R_{ij} \leftarrow R_{ij} \cap \pi_{ij}(R_{ik} \otimes D_k \otimes R_{kj})$$

- Complexity: O(k^3)
- Best-case: O(t)
- Worst-case O(tk)

PC-1

PC-1($\mathcal{R}$)

input: a network $\mathcal{R} = (X, D, C)$.
output: a path consistent network equivalent to $\mathcal{R}$.

1. **repeat**
2. **for** $k \leftarrow 1$ to n
3. **for** $i, j \leftarrow 1$ to n
4. $R_{ij} \leftarrow R_{ij} \cap \pi_{ij}(R_{ik} \bowtie D_k \bowtie R_{kj}) /* \ (Revise - 3((i,j),k))$
5. **endfor**
6. **endfor**
7. **until** no constraint is changed.

Figure 3.10: Path-consistency-1 (PC-1)

- Complexity: $O(n^5 k^5)$
- O(n^3) triplets, each take O(k^3) steps → O($n^3 k^3$)
- Max number of loops: O(n^2 k^2) .

PC-2

PC-3($\mathcal{R}$)

input: a network $\mathcal{R} = (X, D, C)$.
output: $\mathcal{R}'$ a path consistent network equivalent to $\mathcal{R}$.
1. $Q \leftarrow \{(i,k,j) \mid 1 \le i < j \le n, 1 \le k \le n, k \ne i, k \ne j\}$
2. **while** Q is not empty
3. select and delete a 3-tuple (i,k,j) from Q
4. $R_{ij} \leftarrow R_{ij} \cap \pi_{ij}(R_{ik} \bowtie D_k \bowtie R_{kj})$ /* (Revise-3($(i,j),k$))
5. **if** R_{ij} changed then
6. $Q \leftarrow Q \cup \{(l,i,j)(l,j,i) \mid 1 \le l \le n, l \ne i, l \ne j\}$
7. **endwhile**

Figure 3.11: Path-consistency-3 (PC-3)

- Complexity: $O(n^3k^5)$
- Optimal PC-4: $O(n^3k^3)$
- (each pair deleted may add: 2n-1 triplets, number of pairs: O($n^2\ k^2$) → size of Q is O($n^3\ k^2$), processing is O(k^3))

Path-consistency Algorithms

- Apply Revise-3 (O(k^3)) until no change

$$R_{ij} \leftarrow R_{ij} \cap \pi_{ij}(R_{ik} \bowtie D_k \bowtie R_{kj})$$

- Path-consistency (3-consistency) adds binary constraints.
- PC-1: $O(n^5 k^5)$
- PC-2: $O(n^3 k^5)$
- PC-4 optimal: $O(n^3 k^3)$

Local i-Consistency

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

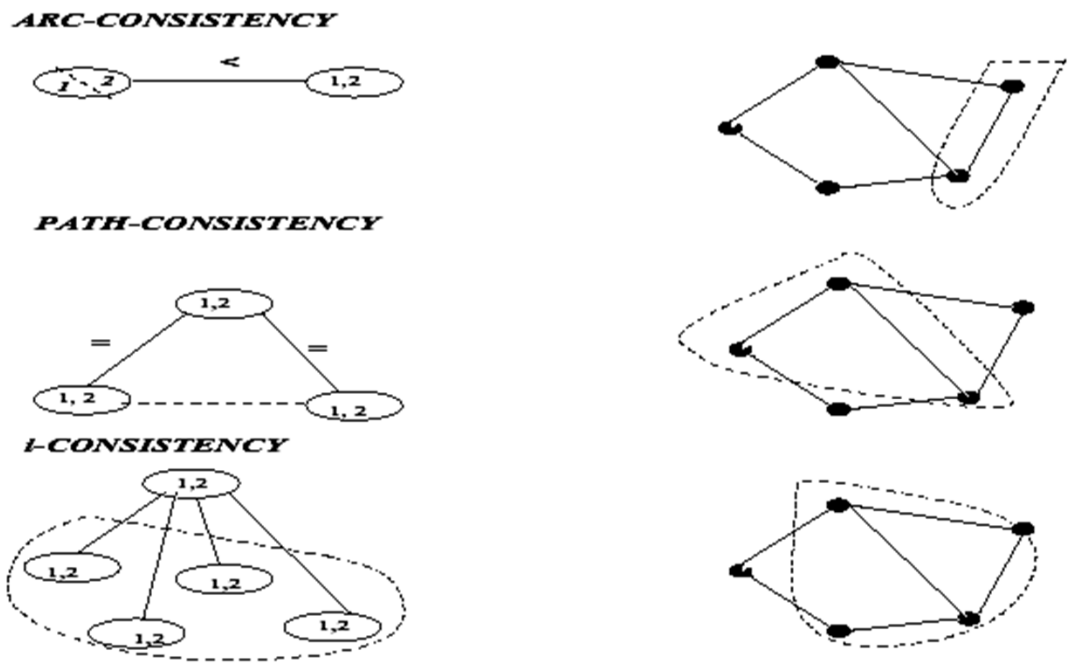
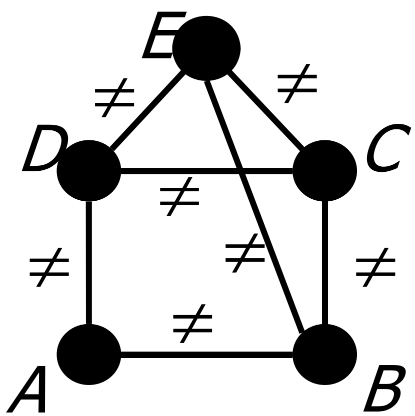


Figure 3.17: The scope of consistency enforcing: (a) arc-consistency, (b) path-consistency, (c) i-consistency

Directional i-Consistency



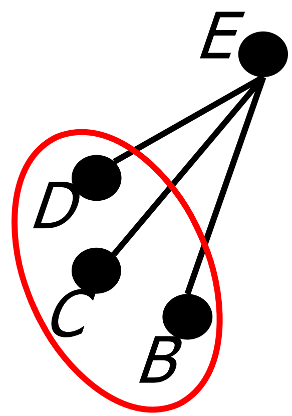
$E: E \neq D, E \neq C, E \neq B$

$D: D \neq C, D \neq A$

$C: C \neq B$

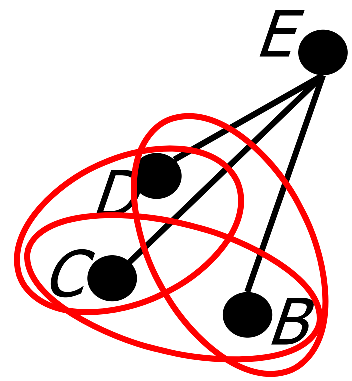
$B: A \neq B$

$A:$



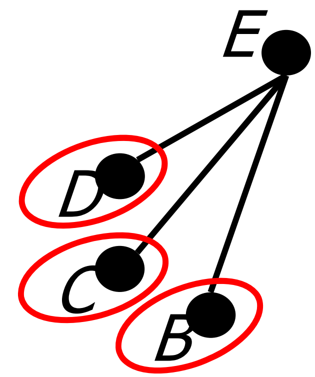
Adaptive

R_{DCB}



d-path

R_{DC}, R_{DB}
R_{CB}



d-arc

R_D
R_C
R_D

Boolean Constraint Propagation

- (A V ¬B) and (B)
  - B is arc-consistent relative to A but not vice-versa
- Arc-consistency by resolution:

  res((A V ¬B),B) = A

Given also (B V C), path-consistency:

res((A V ¬B),(B V C) = (A V C)

Relational arc-consistency rule = unit-resolution

$$A \wedge B \rightarrow G, \neg G, \Rightarrow \neg A \vee \neg B$$

Gausian and Boolean Propagation, Resolution

- Linear inequalities

$$x + y + z \leq 15, z \geq 13 \Rightarrow$$

$$\boxed{x \leq 2, y \leq 2}$$

- Boolean constraint propagation, unit resolution

$$(A \vee B \vee \neg C), (\neg B) \Rightarrow$$

$$(A \vee \neg C)$$

Unit Propagation

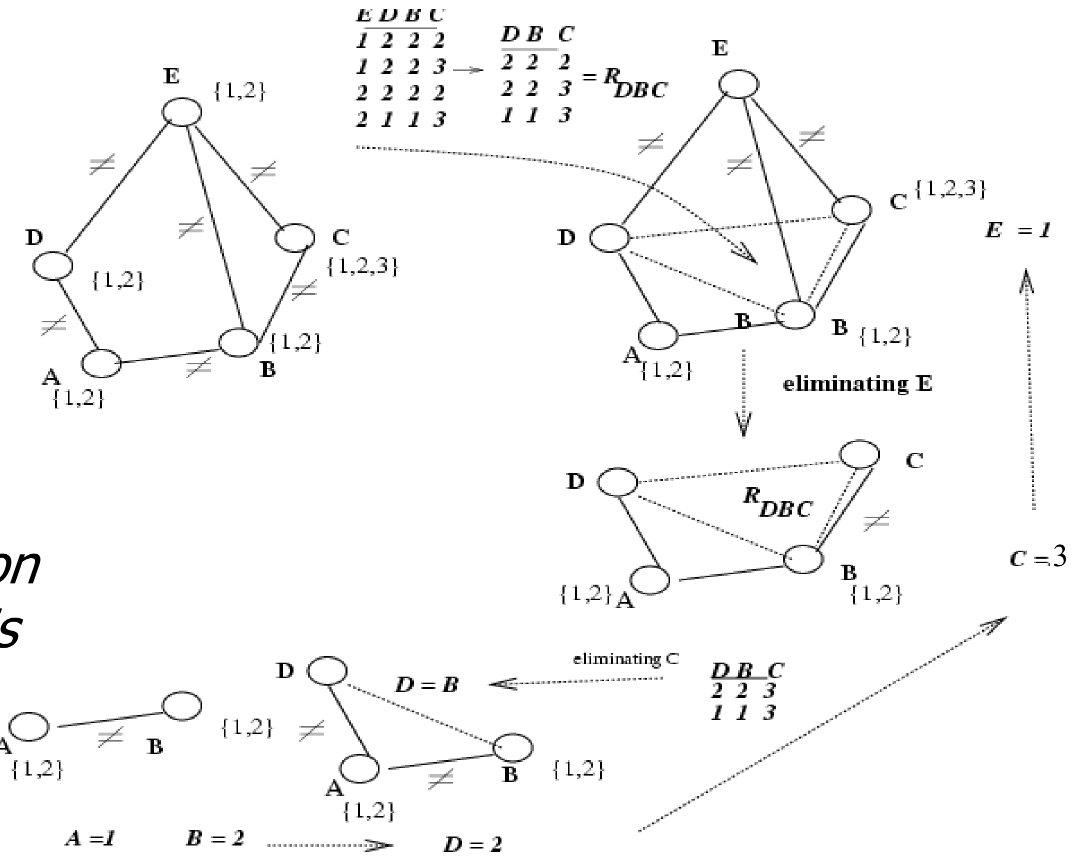
Procedure UNIT-PROPAGATION
Input: A cnf theory, φ, $d = Q_1, ..., Q_n$.
Output: An equivalent theory such that every unit clause does not appear in any non-unit clause.
1. queue = all unit clauses.
2. **while** queue is not empty, do.
3. $T \leftarrow$ next unit clause from Queue.
4. **for** every clause β containing T or $\neg T$
5. **if** β contains T delete β (subsumption elimination)
6. **else**, For each clause $\gamma = resolve(\beta, T)$.
   if γ, the resolvent, is empty, the theory is unsatisfiable.
7. **else**, add the resolvent γ to the theory and delete β.
   if γ is a unit clause, add to Queue.
8. **endfor**.
9. **endwhile**.

Theorem 3.6.1 *Algorithm* UNIT-PROPAGATION *has a linear time complexity.*

Variable Elimination

Eliminate variables one by one: *"constraint propagation"*

Solution generation after elimination is backtrack-free



Road Map

- Graphical models
- Constraint networks Model
- Inference
  - Variable elimination:
  - Tree-clustering
  - Constraint propagation
- Search
- Probabilistic Networks