Directional consistency

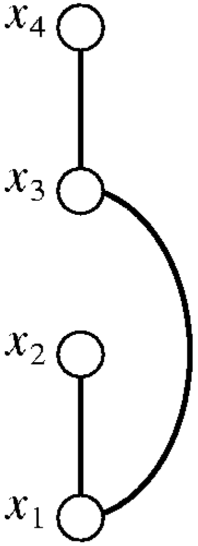
Chapter 4

Backtrack-free search: or

What level of consistency will guarantee global-consistency

Definition 4.1.1 (backtrack-free search) *A constraint network is backtrack-free relative to a given ordering* $d = (x_1, ..., x_n)$ *if for every* $i \leq n$*, every partial solution of* $(x_1, ..., x_i)$ *can be consistently extended to include* x_{i+1}.

Directional arc-consistency:
another restriction on propagation

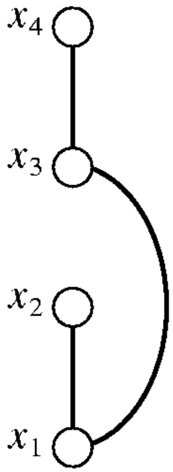


D4={white,blue,black}
D3={red,white,blue}
D2={green,white,black}
D1={red,white,black}
X1=x2, x1=x3,x3=x4

Directional arc-consistency:
another restriction on propagation

Definition 4.3.1 (directional arc-consistency) *A network is* directional-arc-consistent *relative to order* $\boldsymbol{d} = (\boldsymbol{x}_1, ..., \boldsymbol{x}_n)$ *iff every variable* $\boldsymbol{x}_i$ *is arc-consistent relative to every variable* $\boldsymbol{x}_j$ *such that* $\boldsymbol{i} \leq \boldsymbol{j}$.

D4={white,blue,black}
D3={red,white,blue}
D2={green,white,black}
D1={red,white,black}
X1=x2, x1=x3,x3=x4

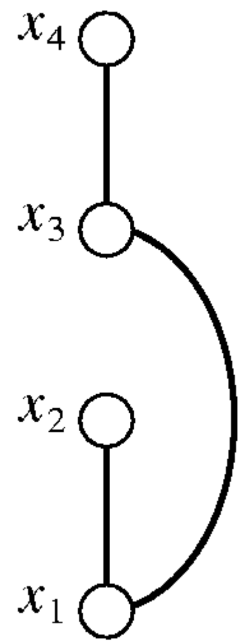


Directional arc-consistency:

another restriction on propagation

- D4={white,blue,black}
- D3={red,white,blue}
- D2={green,white,black}
- D1={red,white,black}
- X1=x2, x1=x3, x3=x4

- After DAC:
- D1= {white},
- D2={green,white,black},
- D3={white,blue},
- D4={white,blue,black}



Algorithm for directional arc-consistency (DAC)

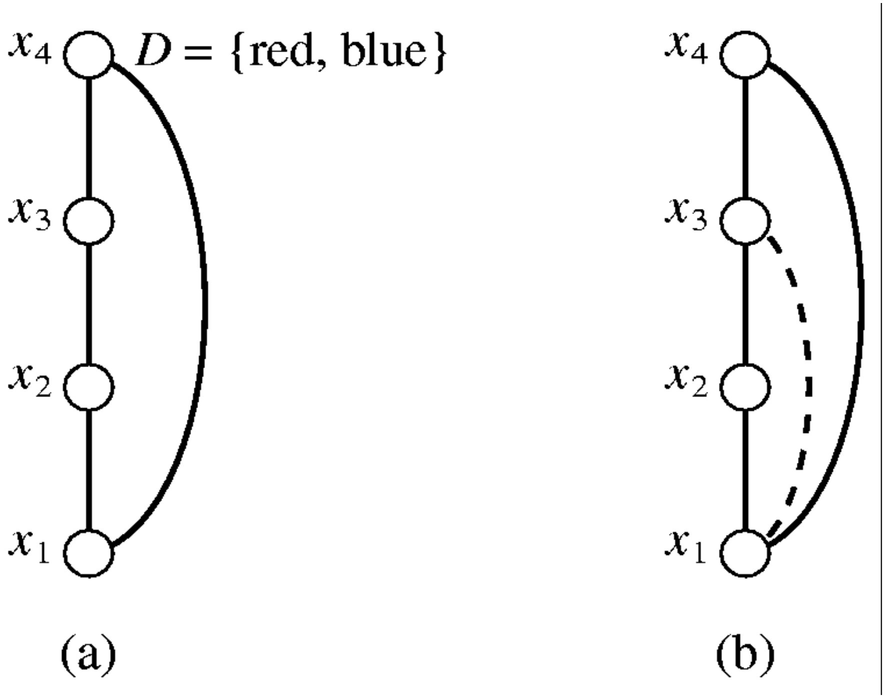
DAC($\mathcal{R}$)
Input:A network $\mathcal{R} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$, its constraint graph G, and an ordering $d = (x_1, ..., x_n)$.
Output: A directional arc-consistent network.

1. **for** $i = n$ to 1 by -1 **do**
2. **for** each $j < i$ s.t. $R_{ji} \in \mathcal{R}$,
3. $D_j \leftarrow D_j \cap \pi_j(R_{ji} \bowtie D_i)$, (this is revise($(x_j), x_i$)).
4. **end-for**

Figure 4.6: Directional arc-consistency (DAC)

- Complexity: $O(ek^2)$

Directional arc-consistencymay not be enough → Directional path-consistency



Definition 4.3.5 (directional path-consistency) *A network* $\mathcal{R}$ *is* directional path-consistent *relative to order* $d = (x_1, ..., x_n)$ *iff for every* $k \geq i, j$, *the pair* $\{x_i, x_j\}$ *is path-consistent relative to* x_k.

Algorithm directional path consistency (DPC)

DPC($\mathcal{R}$)
Input:A binary network $\mathcal{R} = (X, D, C)$ and its constraint graph $G = (V, E)$, $d = (x_1, ..., x_n)$.
Output:A strong directional path-consistent network and its graph $G' = (V, E')$.
Initialize: $E' \leftarrow E$.

1. **for** $k = n$ to 1 by -1 **do**
2. (a) $\forall\ i \leq k$ such that x_i is connected to x_k in the graph, **do**
3. $D_i \leftarrow D_i \cap \pi_i(R_{ik} \bowtie D_k)$ $(Revise((x_i), x_k))$
4. (b) $\forall\ i, j \leq k$ s.t. (x_i, x_k),$(x_j, x_k) \in E'$ **do**
5. $R_{ij} \leftarrow R_{ij} \cap \pi_{ij}(R_{ik} \bowtie D_k \bowtie R_{kj})$ (Revise-3$((x_i, x_j), x_k)$)
6. $E' \leftarrow E' \cup (x_i, x_j)$
7. **endfor**
8. **return** The revised constraint network $\mathcal{R}$ and $G' = (V, E')$.

Theorem 4.3.7 *Given a binary network $\mathcal{R}$ and an ordering d, algorithm DPC generates a largest equivalent, strong, directional-path-consistent network relative to d. The time and space complexity of DPC is $O(n^3k^3)$, where n is the number of variables and k bounds the domain sizes.*

Directional i-consistency

Definition 4.3.8 (directional i-consistency) *A network is* directional i-consistent *relative to order* $\boldsymbol{d} = (\boldsymbol{x}_1, ..., \boldsymbol{x}_n)$ *iff every* $\boldsymbol{i} - 1$ *variables are* $\boldsymbol{i}$*-consistent relative to every variable that succeeds them in the ordering. A network is strong directional* $\boldsymbol{i}$*-consistent if it is directional* $\boldsymbol{j}$*-consistent for every* $\boldsymbol{j} < \boldsymbol{i}$.

Algorithm directional i-consistency

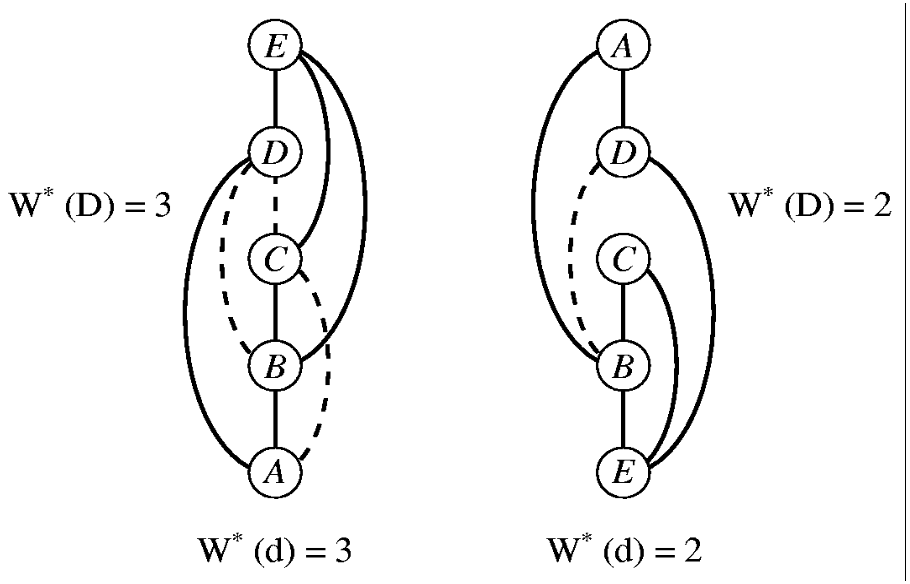
Directional i-consistency $(DIC_i(\mathcal{R}))$
Input: a network $\mathcal{R} = (X, D, C)$, its constraint graph $G = (V, E)$, $d = (x_1, \ldots, x_n)$.
output: A strong directional i-consistent network along d and its graph $G' = (V, E')$.
Initialize: $E' \leftarrow E$, $C' \leftarrow C$.
1. **for** $j = n$ to 1 by -1 **do**
2. let $P = parents(x_j)$.
3. if $|P| < i - 1$ then
4. $Revise(P, x_j)$
5. **else, for** each subset of $i - 1$ variables S, $S \subseteq P$, **do**
6. Revise(S, x_j)
7. **endfor**
8. $C' \leftarrow C' \cup$ all generated constraints.
8. $E' \leftarrow E' \cup \{(x_k, x_m) | x_k, x_m \in P\}$ (connect all parents of x_j)
9. **endfor**.
10. **return** C' and E'.

Figure 4.9: Algorithm directional i-consistency (DIC_i)

Graph aspects of DPC

- DPC recursively connects parents in the ordered graph, yielding:
  - Induced graph
  - Induced-width
  - Min-width ordering
  - Max-cardinality ordering
  - Min-fill ordering
  - Chordal graphs

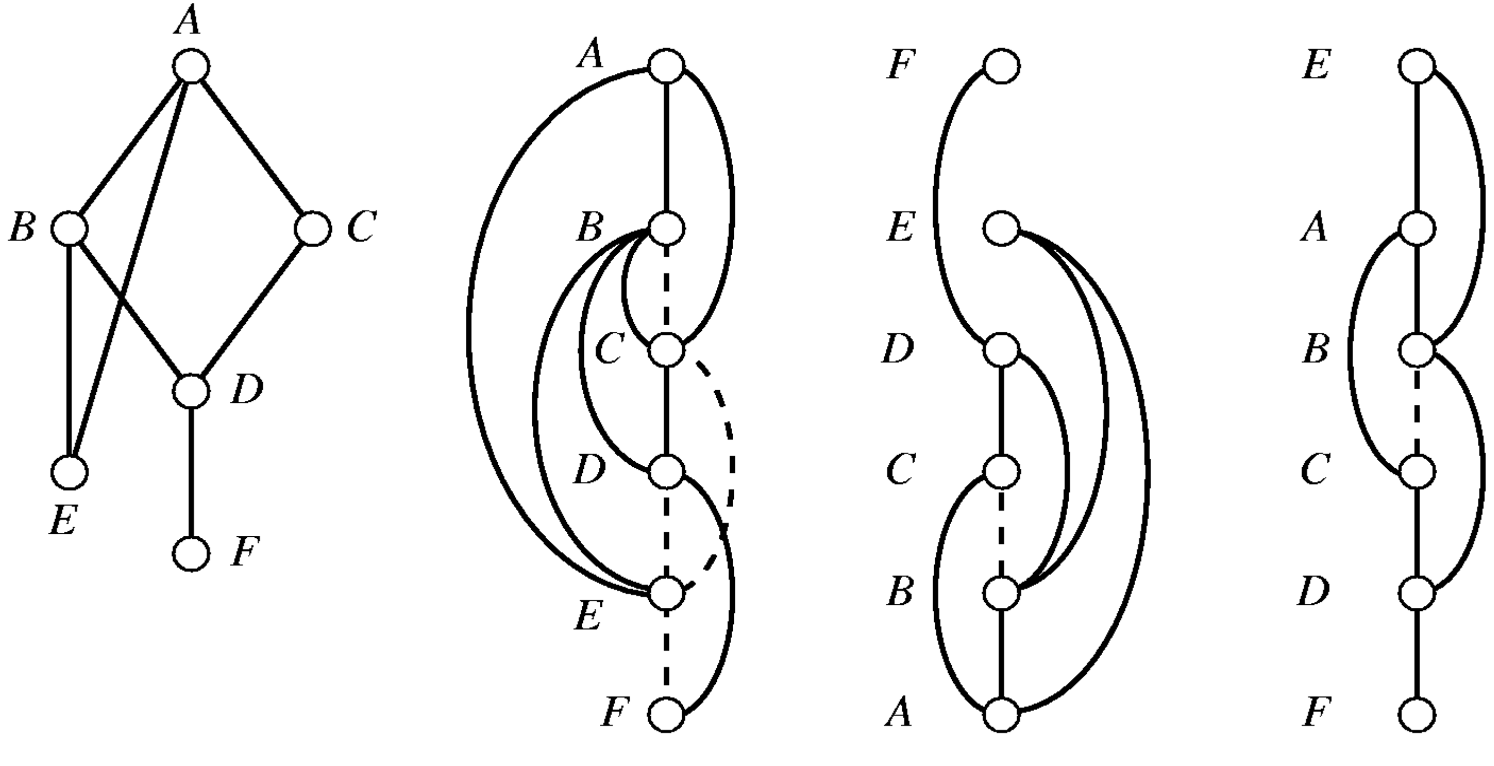
The induced-width



- width: is the max number of parents in the ordered graph
- Induced-width: width of induced graph: recursivlely connecting parents going from last node to first.
- Induced-width w*(d) = the max induced-width over all nodes
- Induced-width of a graph: max w*(d) over all d

Example 4.1: Figure 4.1 presents a constraint graph G over six nodes, along with three orderings of the graph: $d_1 = (F,E,D,C,B,A)$, its reversed ordering $d_2 = (A,B,C,D,E, F)$, and $d_3 = (F,D,C,B,A,E)$. Note that we depict the orderings from bottom to top, so that the first node is at the bottom of the figure and the last node is at the top. The arcs of the graph are depicted by the solid lines. The parents of A along d_1 are $\{B,C,E\}$. The width of A along d_1 is 3, the width of C along d_1 is 1, and the width of A along d_3 is 2. The width of these three orderings are: $w(d_1) = 3$, $w(d_2) = 2$, and $w(d_3) = 2$. The width of graph G is 2.

Induced-width



(a) (b) (c) (d)

Induced-width and DPC

- The induced graph of (G,d) is denoted (G*,d)

- The induced graph (G*,d) contains the graph generated by DPC along d, and the graph generated by directional consistency along d

Refined Complexity using induced-width

Theorem 4.3.11 *Given a binary network $\mathcal{R}$ and an ordering $\mathbf{d}$, the complexity of DPC along $\mathbf{d}$ is $O((w^*(d))^2 \cdot n \cdot k^3)$, where $w^*(\mathbf{d})$ is the induced width of the ordered constraint graph along $\mathbf{d}$.*

Theorem 4.3.13 *Given a general constraint network $\mathcal{R}$ whose constraints' arity is bounded by i, and an ordering $\mathbf{d}$, the complexity of DIC_i along $\mathbf{d}$ is $O(n(w^*(d))^i \cdot (2k)^i)$.* □

- Consequently we wish to have ordering with minimal induced-width
- Induced-width = tree-width
- Finding min induced-width ordering is NP-complete

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**

Figure 4.2: The min-width (MW) ordering procedure

Min-induced-width

MIN-INDUCED-WIDTH (MIW)

input: a graph $G = (V, E)$, $V = \{v_1, ..., v_n\}$
output: An ordering of the nodes $d = (v_1, ..., v_n)$.
1. **for** $j = n$ to 1 by -1 do
2. $r \leftarrow$ a node in V with smallest degree.
3. put r in position j.
4. connect r's neighbors: $E \leftarrow E \cup \{(v_i, v_j) | (v_i, r) \in E, (v_j, r) \in E\}$,
5. remove r from the resulting graph: $V \leftarrow V - \{r\}$.

Figure 4.3: The min-induced-width (MIW) procedure

Min-fill algorithm

- 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)

Cordal graphs and Max-cardinality ordering

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

Example 4.3: We see again that *G* in Figure 4.1(a) is not chordal since the parents of *A* are not connected in the max-cardinality ordering in Figure 4.1(d). If we connect *B* and *C*, the resulting induced graph is chordal.

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. $\quad r \leftarrow$ a node in G that is connected to a largest subset of nodes in positions 1 to $j - 1$, breaking ties arbitrarily.
4. **endfor**

Figure 4.5 The max-cardinality (MC) ordering procedure.

Width vs local consistency: solving trees

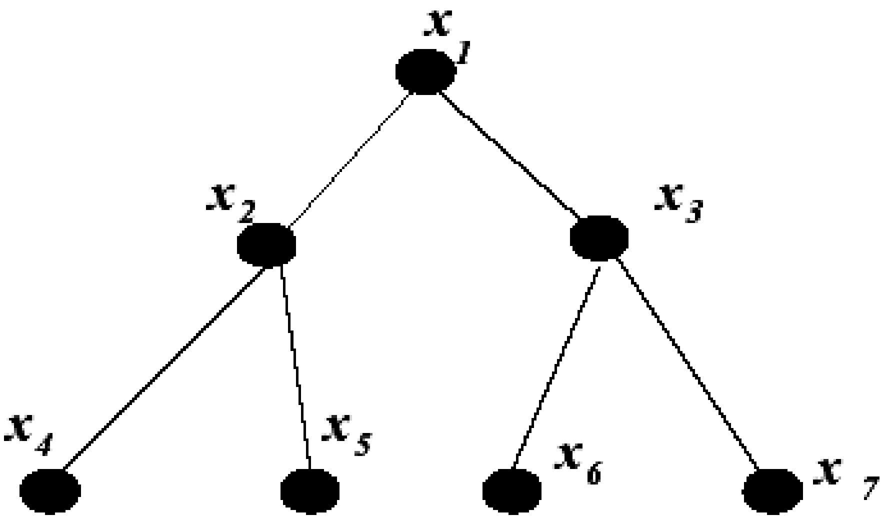


Figure 4.10: A tree network

Theorem 4.4.1 *If a binary constraint network has a width of 1 and if it is arc-consistent, then it is backtrack-free along any width-1 ordering.*

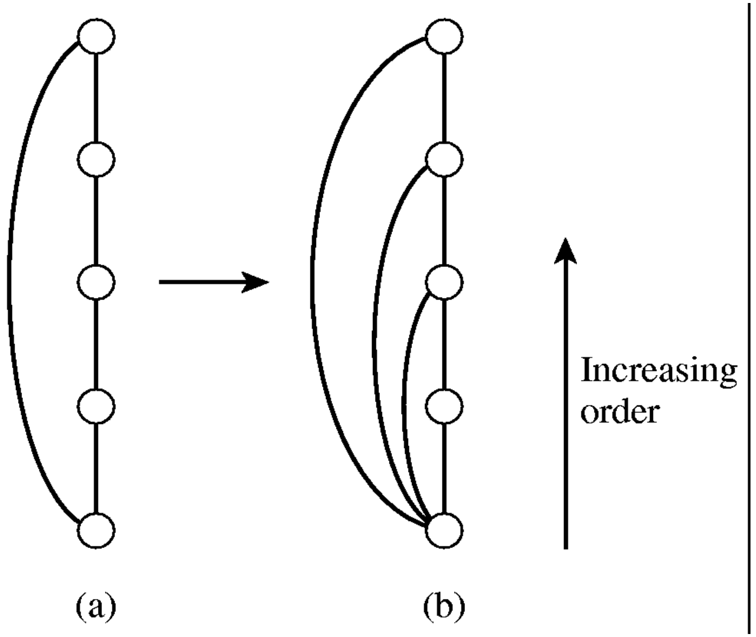
Tree-solving

Tree-solving
Input: A tree network $T = (X, D, C)$.
Output: A backtrack-free network along an ordering d.
1. generate a width-1 ordering, $d = x_1, \ldots, x_n$.
2. **let** $x_{p(i)}$ denote the parent of x_i in the rooted ordered tree.
3. **for** $i = n$ to 1 **do**
4. *Revise* $((x_{p(i)}), x_i)$;
5. **if** the domain of $x_{p(i)}$ is empty, exit. (no solution exists).
6. **endfor**

Figure 4.11: Tree-solving algorithm

$$complexity \quad : O(nk^2)$$

Width-2 and DPC



(a) (b)

Theorem 4.4.3 (Width-2 and directional path-consistency) *If $\mathcal{R}$ is directional arc and path-consistent along $\mathbf{d}$, and if it also has width-2 along $\mathbf{d}$, then it is backtrack-free along $\mathbf{d}$.* □

Width vs directional consistency
(Freuder 82)

Theorem 4.4.5 (Width (i-1) and directional i-consistency) *Given a general network $\mathcal{R}$, its ordered constraint graph along* $\mathbf{d}$ *has a width of* $i-1$ *and if it is also strong directional* $\mathbf{i}$*-consistent, then* $\mathcal{R}$ *is backtrack-free along* $\mathbf{d}$.

Width vs i-consistency

- DAC and width-1
- DPC and width-2
- DIC_i and with-(i-1)
- → backtrack-free representation

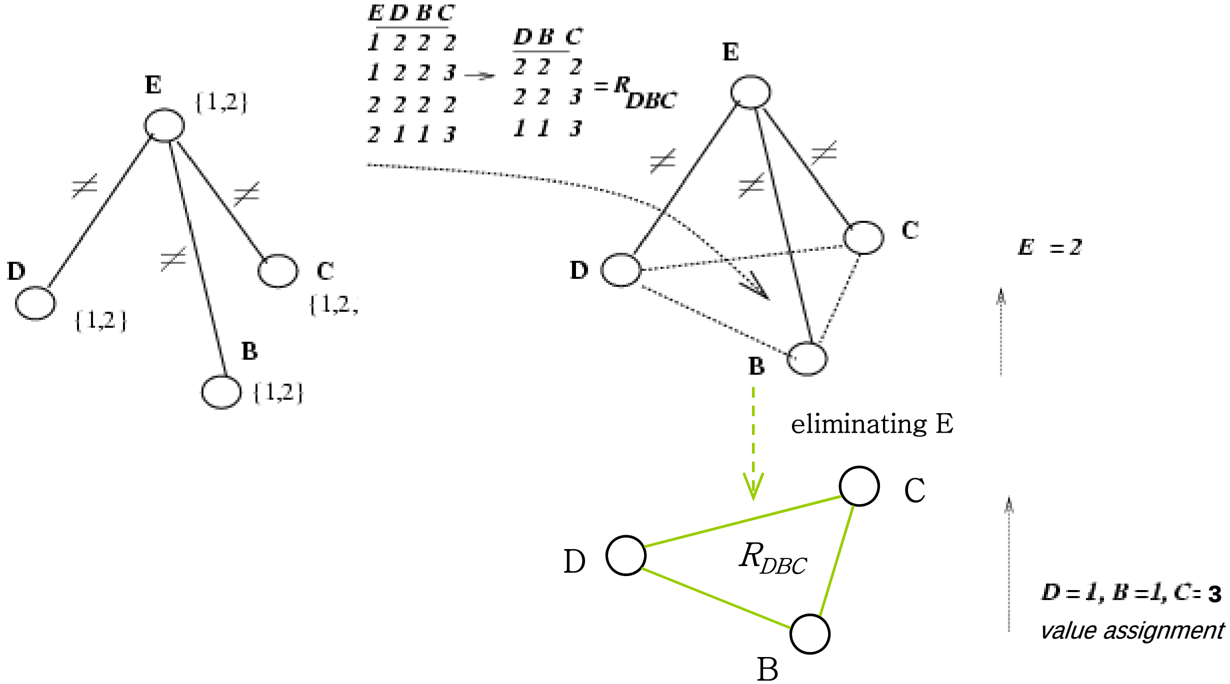
- If a problem has width i-1, will DIC_i make it backtrack-free?
- **Adaptive-consistency**: applies i-consistency when i is adapted to the number of parents

Adaptive-consistency

ADAPTIVE-CONSISTENCY (AC1)
Input: a constraint network $\mathcal{R} = (X, D, C)$, its constraint graph $G = (V, E)$, $d = (x_1, \ldots, x_n)$.
output: A backtrack-free network along d
Initialize: $C' \leftarrow C$, $E' \leftarrow E$
1. **for** $j = n$ to 1 do
2. Let $S \leftarrow parents(x_j)$.
3. $R_S \leftarrow Revise(S, x_j)$ (generate all partial solutions over S that can extend to x_j).
4. $C' \leftarrow C' \cup R_S$
5. $E' \leftarrow E' \cup \{(x_k, x_r) | x_k, x_r \in parents(x_j)\}$ (connect all parents of x_j)
5. **endfor**.

Figure 4.13: Algorithm adaptive-consistency– version 1

The Idea of Elimination



$$R_{DBC} = \prod_{DBC} R_{ED} \bowtie R_{EB} \bowtie R_{EC}$$

Eliminate variable E $\Leftrightarrow$ join and project

Adaptive-consistency, bucket-elimination

ADAPTIVE-CONSISTENCY (AC)

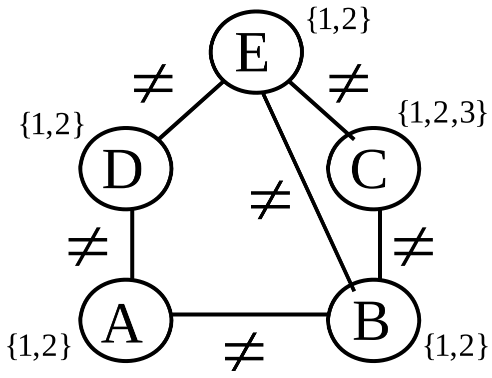
Input: a constraint network $\mathcal{R}$, an ordering $d = (x_1, \ldots, x_n)$
output: A backtrack-free network, denoted $E_d(\mathcal{R})$, along d, if the empty constraint was not generated. Else, the problem is inconsistent

1. Partition constraints into $bucket_1, \ldots, bucket_n$ as follows:
   for $i \leftarrow n$ **downto** 1, put in $bucket_i$ all unplaced constraints mentioning x_i.
2. **for** $p \leftarrow n$ **downto** 1 **do**
3. **for** all the constraints $R_{S_1}, \ldots, R_{S_j}$ in $bucket_p$ **do**
4. $A \leftarrow \bigcup_{i=1}^{j} S_i - \{x_p\}$
5. $R_A \leftarrow \Pi_A(\bowtie_{i=1}^{j} R_{S_i})$
6. **if** R_A is not the empty relation **then** add R_A to the bucket of the latest variable in scope A,
7. **else** exit and return the empty network
8. **return** $E_d(\mathcal{R}) = (X, D, bucket_1 \cup bucket_2 \cup \cdots \cup bucket_n)$

Figure 4.14: Adaptive-Consistency as a bucket-elimination algorithm

Bucket Elimination

Adaptive Consistency (Dechter and Pearl, 1987)



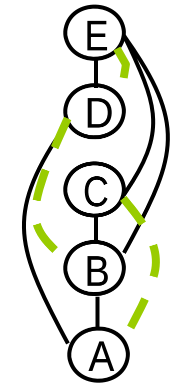
$Bucket(E):\ \mathrm{E} \neq \mathrm{D},\ \mathrm{E} \neq \mathrm{C},\ \mathrm{E} \neq \mathrm{B}$

$Bucket(D):\ \mathrm{D} \neq \mathrm{A}\ //\ R_{DCB}$

$Bucket(C):\ \mathrm{C} \neq \mathrm{B}\ //\ R_{ACB}$

$Bucket(B):\ \mathrm{B} \neq \mathrm{A}\ //\ R_{AB}$

$Bucket(A):\ R_{A}$



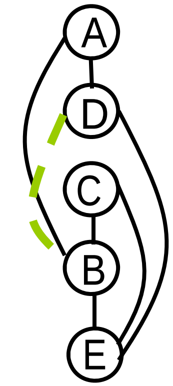
$Bucket(A):\ \mathrm{A} \neq \mathrm{D},\ \mathrm{A} \neq \mathrm{B}$

$Bucket(D):\ \mathrm{D} \neq \mathrm{E}\ //\ R_{DB}$

$Bucket(C):\ \mathrm{C} \neq \mathrm{B},\ \mathrm{C} \neq \mathrm{E}$

$Bucket(B):\ \mathrm{B} \neq \mathrm{E}\ //\ R^{D}_{BE}, R^{C}_{BE}$

$Bucket(E):\ //\ R_{E}$



Complexity : $O(n\ exp(w^*(d)))$,

w^*(d) - ***induced widthalong ordering d***

Properties of bucket-elimination (adaptive consistency)

- **Adaptive consistency generates a constraint network that is backtrack-free (can be solved without dead-ends).**

- **The time and space complexity of adaptive consistency along ordering *d* is** $O(n\,(2k)^{w^*+1}),\ O(n\,(k)^{w^*+1})$ **respectively, or O(r k^(w*+1)) when r is the number of constraints.**

- **Therefore, problems having bounded induced width are tractable (solved in polynomial time)**

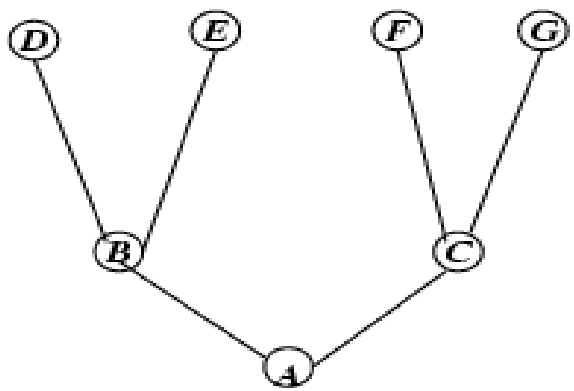
- **Special cases: *trees* (w*=1), *series-parallel networks* (w*=2), and in general *k-trees* (w*=k).**

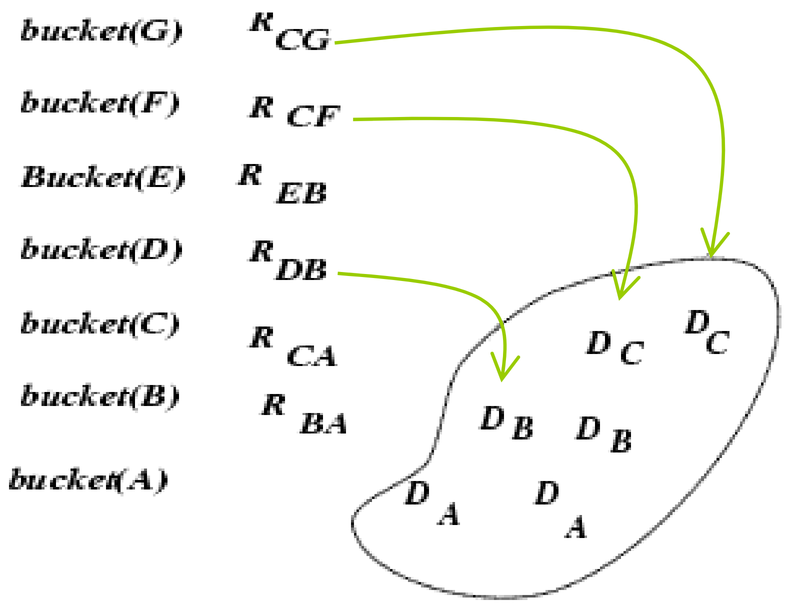
Back to Induced width

- Finding minimum-w* ordering is NP-complete (Arnborg, 1985)
- Greedy ordering heuristics: *min-width, min-degree, max-cardinality* (Bertele and Briochi, 1972; Freuder 1982), Min-fill.

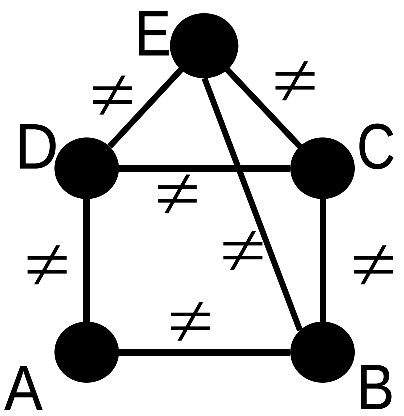
Solving Trees
(Mackworth and Freuder, 1985)

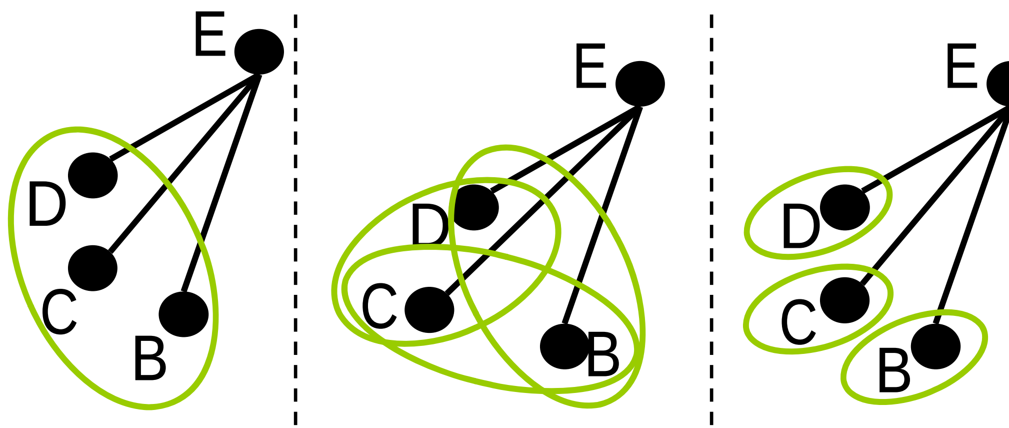
Adaptive consistency is linear for trees and equivalent to enforcing directional arc-consistency (recording only unary constraints)





Summary: directional i-consistency





E : $\mathbf{E \neq D, E \neq C, E \neq B}$

D : $\mathbf{D \neq C, D \neq A}$

C : $\mathbf{C \neq B}$

B : $\mathbf{A \neq B}$

A :

Adaptive	d-path	d-arc
R_{DCB}	R_{DC}, R_{DB} R_{CB}	R_D R_C R_D

Variable Elimination

Eliminate variables one by one: “constraint propagation”

Solution generation after elimination is backtrack-free