Directional Consistency

Chapter 4 @ Constraint Processing by Rina Dechter

Summary: previous chapter

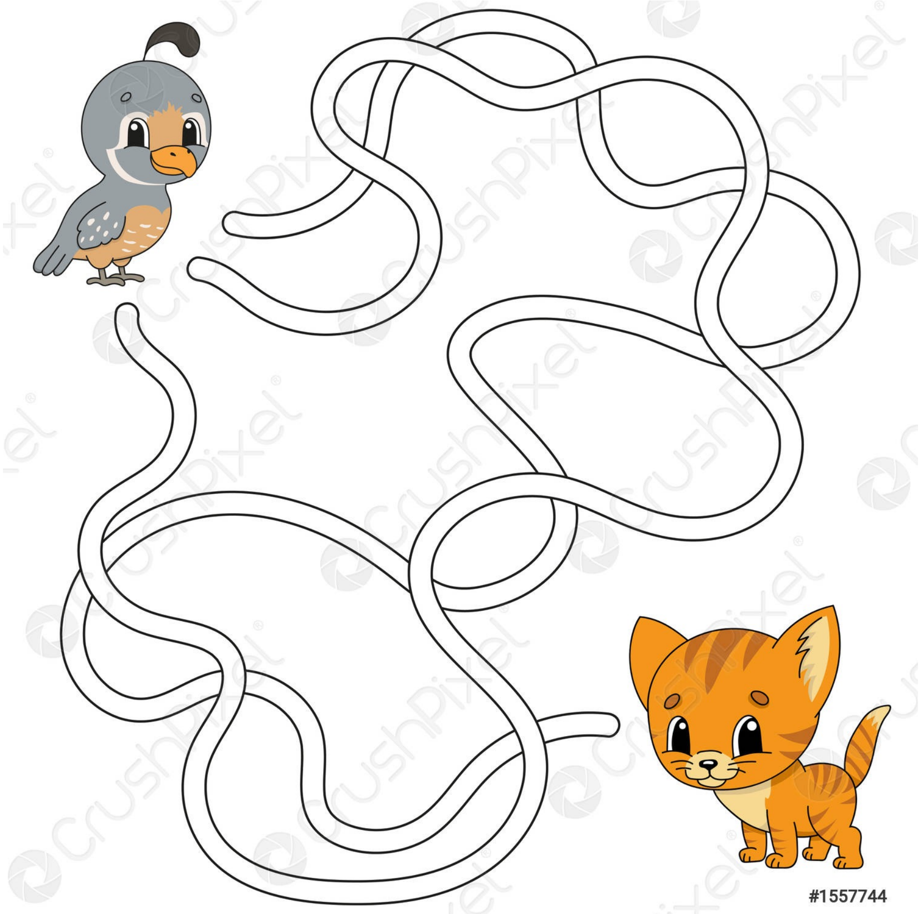
- Basic inference algorithms
  - Arc-, path- and i-consistency
- These algorithms are consistency enforcing and infer new constraints, thus changing the structure of the problem
  - Arc-consistency restricts the domains of variables
  - Path-consistency restricts and adds constraints on pairs of variables
  - i-consistency enforces constraints of arity i-1

Motivation: this chapter

- How do we explain how people perform so well on tasks that are theoretically intractable?
  - We may assume that intelligent behavior is actually grounded in approximation methods that are based on easy-to-solve models
  - i.e. people intuitively transform hard tasks into more manageable tasks
- How does this apply to the AI field?
  - Approximate solutions
  - Heuristics
- When is a problem said to be easy?
  - When it can be solved in polynomial time
  - i.e. backtrack free in the context of CSP

Related puzzle ☺

- How to find the way?



Backtrack-free search

(backtrack-free search)

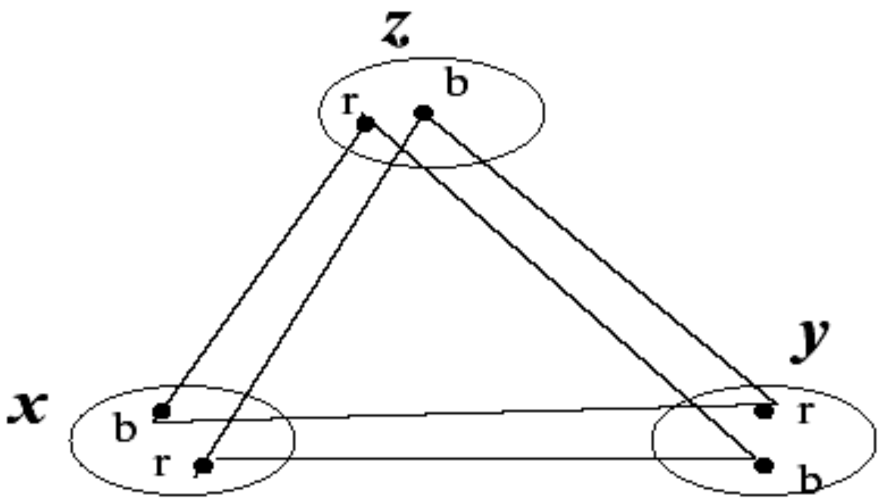
A constraint network is backtrack-free relative to a given ordering $d = (x_1, \ldots, x_n)$ if for every $i \leq n$, every partial solution of $(x_1, \ldots, x_i)$ can be consistently extended to include x_{i+1}. ●

- GOAL: Determine the amount of inference that can guarantee a backtrack- free solution

Tractable classes of CSP

- Tractability by restricted structure
  - Based on reasoning over the constraint graph
  - Independent of the actual constraint relations
  - The focus of this chapter
- Tractability by restricted constraint relations
  - Identify classes that are tractable thanks to special properties of the constraint relation

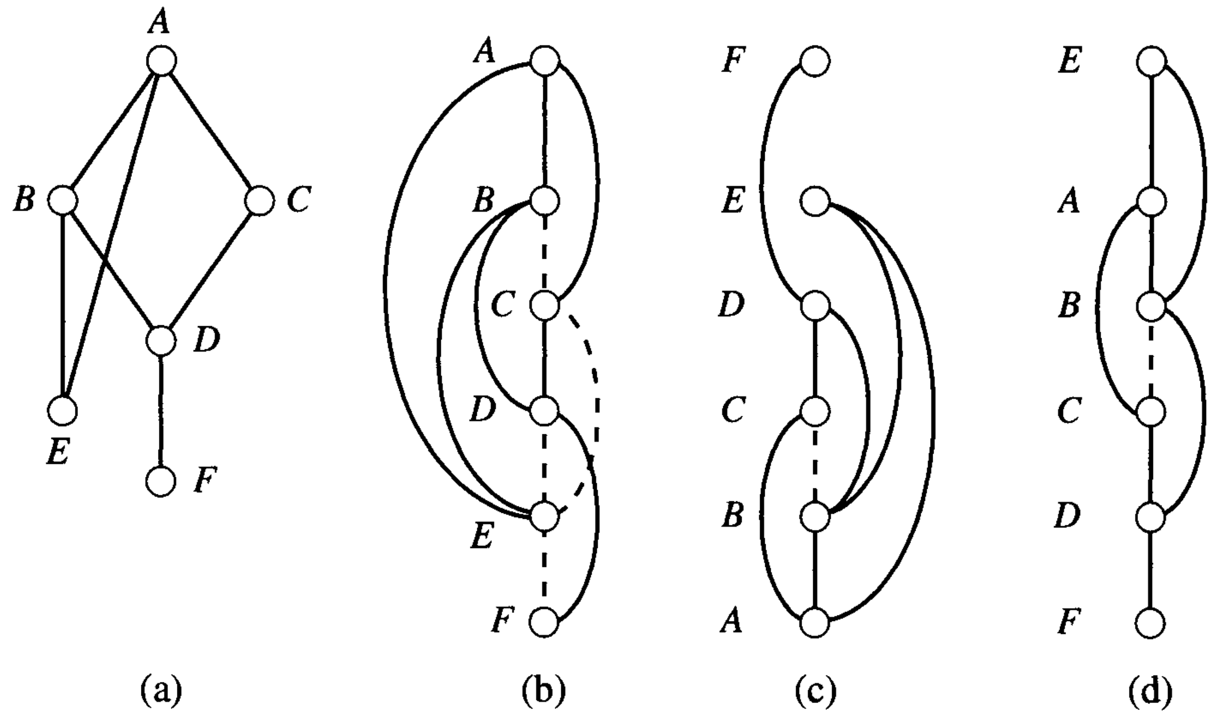
Recap: AC-3 and PC-3



Ordered graph

- Undirected graph G=(V,E)
  - $V=\{v_1,\ldots,v_n\}$ set of nodes
  - E set of edges/arcs over V
- Ordered graph (G,d)
  - $d=(v_1,\ldots,v_n)$ ordering of the nodes
- How to build an ordered graph
  - The nodes are depicted from bottom to top
  - Parents of a node v: nodes adjacent to v that precede v in d
  - Width of a node: number of parents
  - Width of an ordering w(d): *maximum* width over all nodes
  - Width of a graph: *minimum* width over all the orderings of the graph

Example

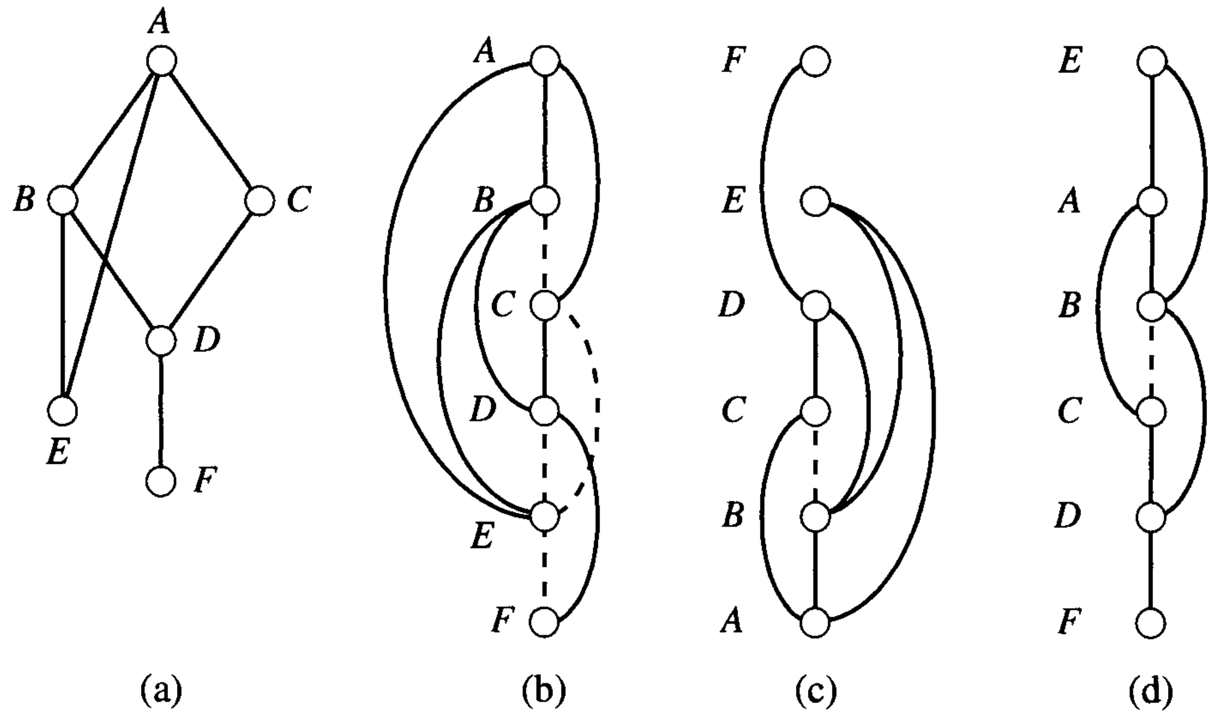


(a) Graph G, and three orderings of the graph: (b) $d_1 = (F, E, D, C, B, A)$, (c) $d_2 = (A, B, C, D, E, F)$, and (d) $d_3 = (F, D, C, B, A, E)$. Broken lines indicate edges added in the induced graph of each ordering.

- Parents of A along d1?
- Width of A along d1?
- Width of C along d1?
- Width of A along d3?
- $w(d_1)$? $w(d_2)$? $w(d_3)$?
- Width of graph G?

You have 5 minutes!

Example



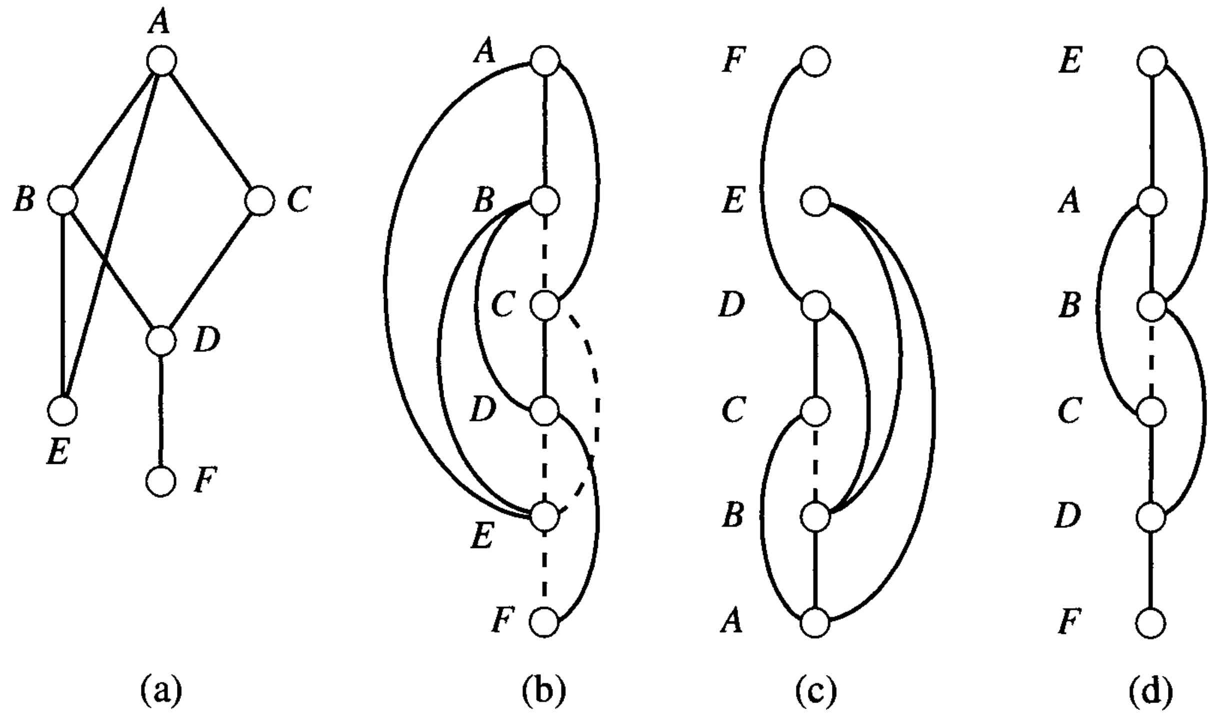
(a) Graph G, and three orderings of the graph: (b) $d_1 = (F, E, D, C, B, A)$, (c) $d_2 = (A, B, C, D, E, F)$, and (d) $d_3 = (F, D, C, B, A, E)$. Broken lines indicate edges added in the induced graph of each ordering.

- Parents of A along d_1? {B,C,E}
- Width of A along d_1? 3
- Width of C along d_1? 1
- Width of A along d_3? 2
- $w(d_1)$? 3
- $w(d_2)$? 2
- $w(d_3)$? 2
- Width of graph G? 2

(G,d): Induced width and induced graph

- The induced graph of (G,d) is an ordered graph (G*,d)
- G* is obtained from G as follows
  - Nodes of G processed from last to first (top to bottom) along d
  - When a node is processed, all of its parents are connected
- Induced width of (G,d), w*(d), is the width of (G*,d)
- Induced width of a graph, w*, is the minimal induced width over all its orderings

Example



(a) Graph G, and three orderings of the graph: (b) $d_1 = (F, E, D, C, B, A)$, (c) $d_2 = (A, B, C, D, E, F)$, and (d) $d_3 = (F, D, C, B, A, E)$. Broken lines indicate edges added in the induced graph of each ordering.

- G* includes the broken lines
- Induced width of B along d_1? 3
- Induced width along d_1? 3
- Induced width along d_2? 2
- Induced width along d_3? 2
- Induced width of G? $w^*(G) = 2$

Observations

- A width-1 graph cannot have a cicle
  - Otherwise at least one node in the cicle would have two parents
- Given an ordering with width 1, the graph has induced width 1
- A graph is a tree iff it has induced width of 1

Greedy algorithms for induced width

- How to find a minimum(-induced) width ordering of a graph?

- Algorithm Min-Width (finds a minimum-width)
- Algorithm Min-Induced-Width (finding minimum is NP-complete; greedy algorithm)

Min-Width

MIN-WIDTH (MW)

Input: A graph $G = (V, E)$, $V = \{v_1, \ldots, v_n\}$.

Output: A min-width ordering of the nodes $d = (v_1, \ldots, v_n)$.

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 $G \leftarrow G - r$.
   $\quad$ (Delete from V node r and from E all its adjacent edges)
4. **endfor**

- Finds a minimum-width ordering of a graph
- Variable with *minimum number of neighbors* put last in the ordering
  - Variable and adjacent edges are then removed from graph
- Could generate d_2

Min-induced-width *(greedy)*

MIN-INDUCED-WIDTH (MIW)

Input: A graph $G = (V, E)$, $V = \{v_1, \ldots, v_n\}$.

Output: An ordering of the nodes $d = (v_1, \ldots, v_n)$.

1. **for** $j = n$ to 1 by -1 do
2. $\quad r \leftarrow$ a node in V with smallest degree.
3. $\quad$ Put r in position j.
4. $\quad$ Connect r's neighbors: $E \leftarrow E \cup \{(v_i, v_j) | (v_i, r) \in E, (v_j, r) \in E\}$.
5. $\quad$ Remove r from the resulting graph: $V \leftarrow V - \{r\}$.

- Variable r *with minimum number of neighbors* put last in the ordering
  - Connect r neighbors
  - Variable is then removed from graph
- Could generate d_2

Directional local consistency

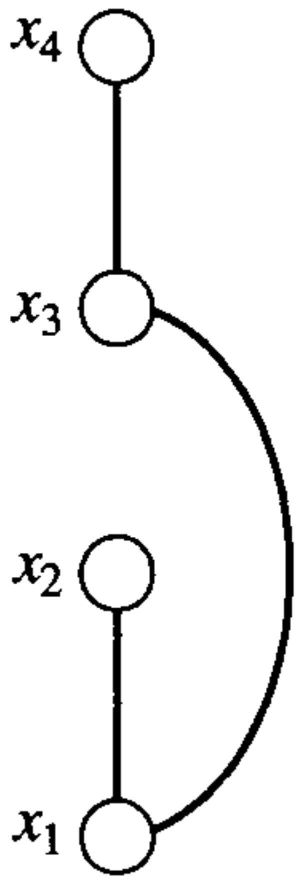
- Back to the primary target:

Determine the amount of inference that can guarantee a backtrack-free solution

Previous chapter: bound the number of variables involved in the inference

Example

- Consider task of applying search to the ordered graph
  - Ensure no dead ends with order d=(x_1,x_2,x_3,x_4)
  - Guarantee that any assignment to x1 has corresponding consistent value in x_2 and x_3
    - Make x_1 arc-consistency with x_2 and x_3
  - Guarantee that any assignment to x_3 has corresponding consistent value in x_4
    - Make x_3 arc-consistency with x_4
  - Arc-consistency only relevant in the search direction



Directional arc-consistency (II)

(directional arc-consistency)

A network is *directional arc-consistent* relative to order $d = (x_1, \ldots, x_n)$ iff every variable x_i is arc-consistent relative to every variable x_j such that $i \leq j$. •

Directional arc-consistency (II)

DAC($\mathcal{R}$)

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

Output: A directional arc-consistent network.

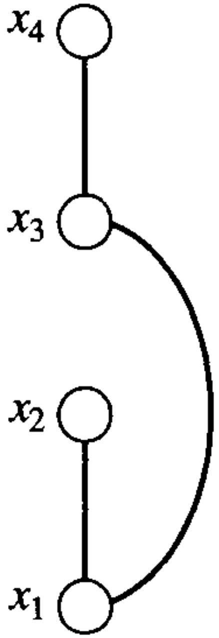
1. **for** $i = n$ to 1 by -1 **do**
2. **for** each $j < i$ such that $R_{ji} \in \mathcal{R}$, do
3. $D_j \leftarrow D_j \cap \pi_j (R_{ji} \bowtie D_i)$, (this is REVISE($(x_j)$, x_i)).
4. **endfor**

- Process variables in reverse order of d
- When processing x_i, reduce domain D_j for each relation R_{ji}
- Question: how many times is processed each constraint?

DAC: example (I)

You have
5 minutes!

$$D_1 = \{red, white, black\}$$
$$D_2 = \{green, white, black\}$$
$$D_3 = \{red, white, blue\}$$
$$D_4 = \{white, blue, black\}$$
$$R_{12}: \quad x_1 = x_2$$
$$R_{13}: \quad x_1 = x_3$$
$$R_{34}: \quad x_3 = x_4$$



- Ordering d=(x_1,x_2,x_3,x_4)
- Apply DAC!
- Process x_4, revise x_3, delete red from D_3
  - D_3 = {white,blue}
- Process x_3, revise x_1, delete red+black from D_1
  - D_1 = {white}
- Process x_2, revise x_1, nothing changes
- D_1={white}, D_2={green, white,black}, D_3={white, blue},D_4={white,blue,black}

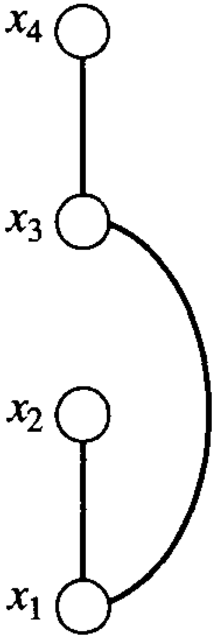
DAC: example (II)

You have
2 minutes!

$R_{12}: \; x_1 = x_2$

$R_{13}: \; x_1 = x_3$

$R_{34}: \; x_3 = x_4$



- D_1={white}, D_2={green, white,black}, D_3={white, blue}, D_4={white,blue,black}
- Is this (full) arc consistency?
- No!
- x_3 is not arc-consistent with x_1
- What if we assign values with ordering (x_1,x_2,x_3,x_4)?
  - Solution $x_1 = x_2 = x_3 = x_4$ = white!

DAC: another example

- Variables x_1,x_2,x_3
- Domains = {red,blue}
- Not equal constraints R_{ij}: $xi \neq xj$, $i \neq j$

- For any ordering, the network is full arc consistent!
  - And directional arc consistent by definition
- Consistent partial assignment: x_1=red, x_2=blue
  - But no consistent assignment to x_3…
- DAC may not be enough… directional path consistency!

Directional path-consistency

- DAC can be extended to directional path consistency
  - And directional i-consistency

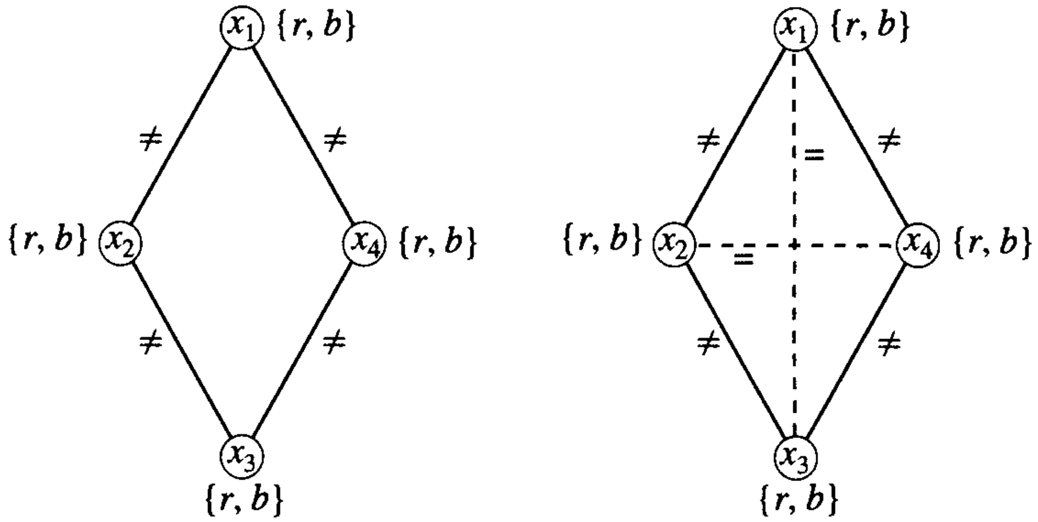
(directional path-consistency)

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

Directional path-consistency: example

- Variables x_1,x_2,x_3,x_4
- Domains = {red,blue}
- Ordering = (x_1,x_2,x_3,x_4)
- Constraints:

$x_1 \neq x_2,\ x_2 \neq x_3,\ x_3 \neq x_4,\ x_4 \neq x_1$



- This network is arc-consistent but not path-consistent
- Enforcing path consistency adds two equal constraints

Directional path-consistency: algorithm

DPC($\mathcal{R}$)

Input: A binary network $\mathcal{R} = (X, D, C)$ and its constraint graph $G = (V, E)$,
$d = (x_1, \ldots, x_n)$.

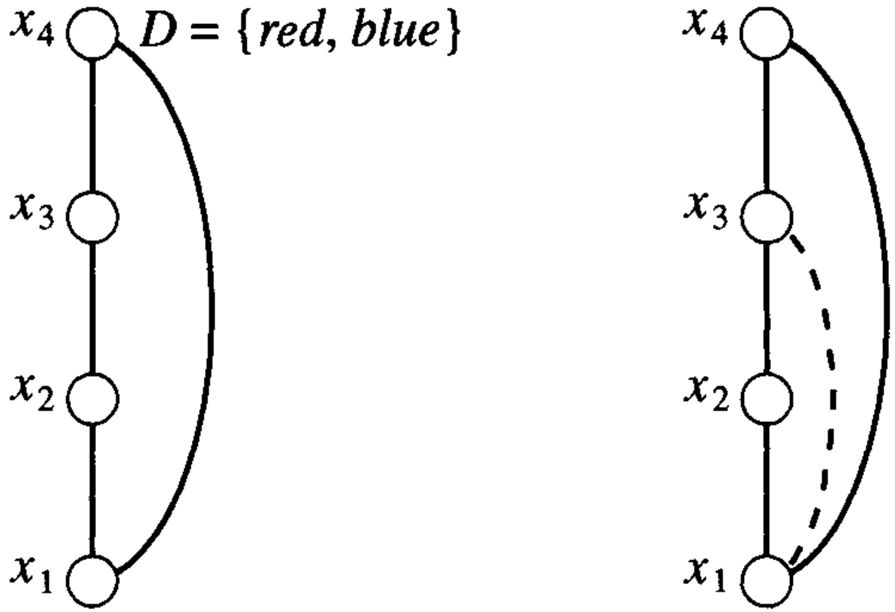
Output: A strong directional path-consistent network and its graph $G' = (V, E')$.

Initialize: $E' \leftarrow E$.

1. **for** $k = n$ to 1 by -1 **do**
2. (a) $\forall\, i \le k$ 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) $\forall\, i,j \le k$ such that $(x_i, x_k), (x_j, x_k) \in E'$ **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 $G' = (V, E')$.

Directional path-consistency: example

- Ordered graph d=(x_1,x_2,x_3,x_4)
- DPC adds only constraint x_1=x_3
  - This allows a solution to be assembled along order d without encountering dead-ends



Directional i-consistency: definition

(directional *i*-consistency)

A network is *directional i-consistent* relative to order $d = (x_1, \ldots, x_n)$ iff every $i - 1$ variables are i-consistent relative to every variable that succeeds them in the ordering. A network is *strong directional i-consistent* if it is directional j-consistent for every $j \leq i$. •

Directional i-consistency: algorithm

Directional *i*-consistency (DIC$_i$($\mathcal{R}$))

Input: A network $\mathcal{R} = (X, D, C)$, its constraint graph $G = (V, E)$, $d = (x_1, \ldots, x_n)$.

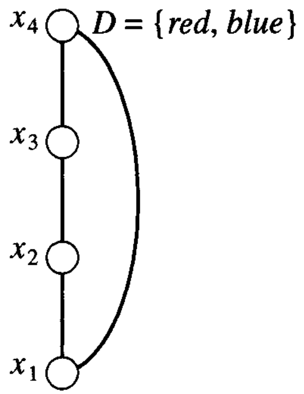
Output: A strong directional *i*-consistent network along d and its graph $G' = (V, E')$.

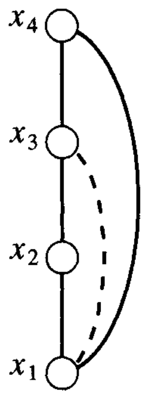
Initialize: $E' \leftarrow E$, $C' \leftarrow C$.

1. **for** $j = n$ to 1 by -1 **do**
2. **let** $P = parents(x_j)$.
3. **if** $|P| \leq i - 1$ 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.
9. $E' \leftarrow E' \cup \{(x_k, x_m) | x_k, x_m \in P\}$ (connect all parents of x_j)
10. **endfor**
11. **return** C' and E'.

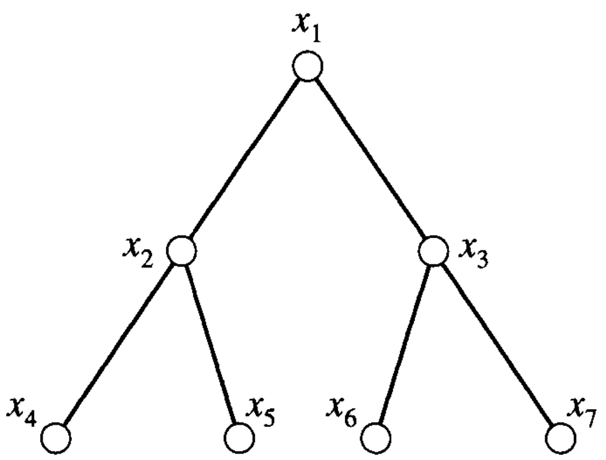
Width vs Local consistency

- Example: DPC changed the network so that a solution is backtrack free
- Easy to find examples where this does not happen
- Question: how to identify, in advance, the level of consistency suficient for generating a backtrack-free representation?





Solving trees: case of width 1



TREE-SOLVING

Input: A tree network $T = (X, D, C)$.

Output: A backtrack-free network along an ordering d.

Generate a width-1 ordering, $d = x_1, \ldots, x_n$ along a rooted tree.
let $x_{p(i)}$ denote the parent of x_i in the rooted ordered tree.
for $i = n$ to 1 **do**
    REVISE $((x_{p(i)}), x_i)$;
    if the domain of $x_{p(i)}$ is empty, exit (no solution exists).
endfor

- Width with ordering $x_1, x_2, \ldots, x_7$? 1
- Width with ordering $x_7, x_6, \ldots, x_1$? 2

Solving trees: case of width 1

(width 1 and directional arc-consistency)

Let d be a width-1 ordering of a constraint tree T. If T is directional arc-consistent relative to d, then the network is backtrack-free along d.

Solving width-2 problems

(width 2 and directional path-consistency)

If $\mathcal{R}$ is directional arc- and path-consistent along d, and if it also has width 2 along d, then it is backtrack-free along d.

- How to identify width-2 problems?
  - Use MIN-INDUCED-WIDTH algorithm!

Solving width-i problems

(Width $i - 1$ and directional i-consistency)

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

Summary

- Introduced the notion of bounded directional consistency algorithms
  - Directional arc-, path- and i-consistency
- These inference algorithms are incomplete but can sometimes decide inconsistency
  - Are mainly designed as preprocessing algorithms to be use before backtracking search
  - Can also be interleaved with search (next chapter)
- Established a relationship between induced width and consistency levels that guaranteed a backtrack-free solution
  - If a problem has width i and it is (i+1)-consistent, then it is backtrack-free


That's all Folks!