CompSci 275, CONSTRAINT Networks

Rina Dechter, Fall 2022

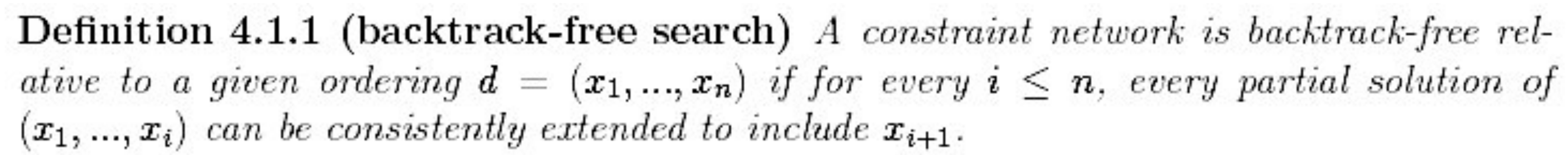
Directional consistency
Chapter 4

Outline

- Directional Arc-consistency algorithms
- Directional Path-consistency and directional i-consistency
- Greedy algorithms for induced-width
- Width and local consistency
- Adaptive-consistency and bucket-elimination

Backtrack-free search: or
What level of consistency will guarantee global-consistency

Let’s explore how we can make a problem backtrack-free with a minimal amount of effort



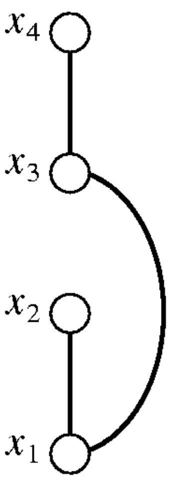
Definition 4.1.1 (backtrack-free search) *A constraint network is backtrack-free relative to a given ordering* $\mathbf{d} = (x_1, ..., x_n)$ *if for every* $i \leq n$*, every partial solution of* $(x_1, ..., x_i)$ *can be consistently extended to include* x_{i+1}.

Backtrack free and queries:
Consistency,
All solutions
Counting
optimization

Directional arc-consistency: another restriction on propagation

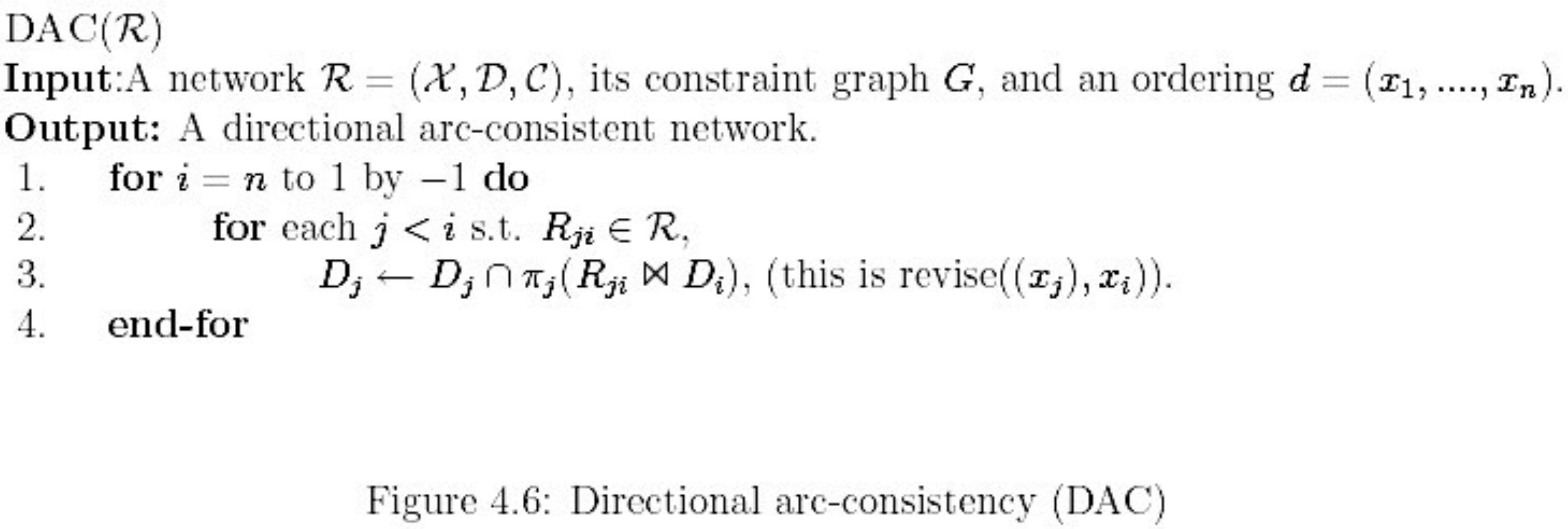
Example 4.3.2 Assume that the constraints and the domains of the problem in Figure 4.5 are specified below.

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



Definition 4.3.1 (directional arc-consistency) *A network is* directional-arc-consistent *relative to order* $d = (x_1, ..., x_n)$ *iff every variable* x_i *is arc-consistent relative to every variable* x_j *such that* $i \leq j$.

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.

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

Figure 4.6: Directional arc-consistency (DAC)

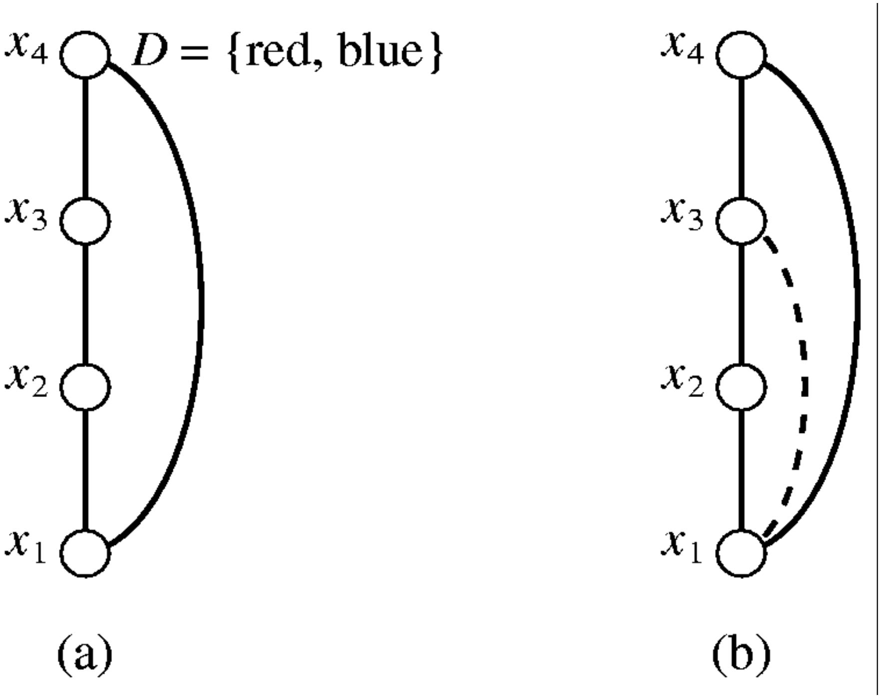
- Complexity:

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

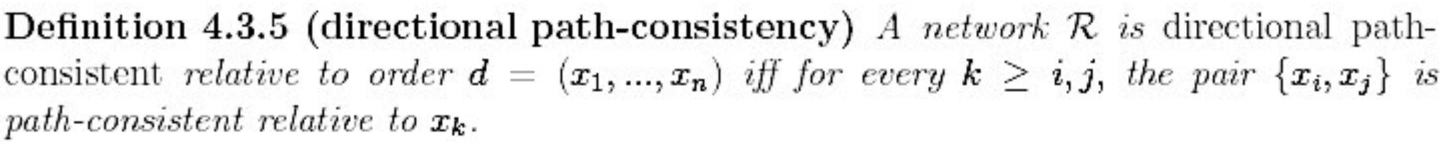
Not equal constraints

2 colors in domains

Is it arc-consistent?

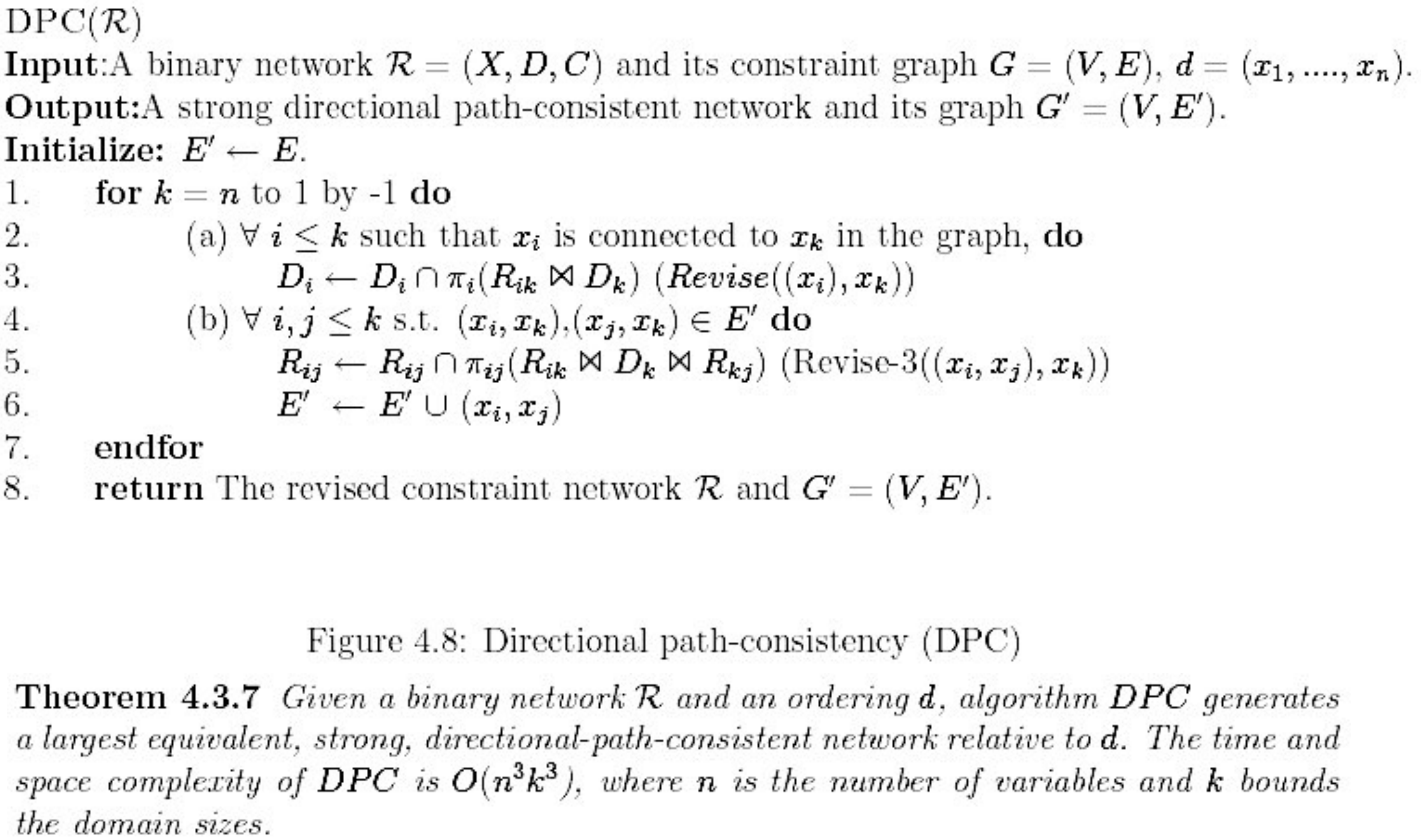


(a) (b)



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

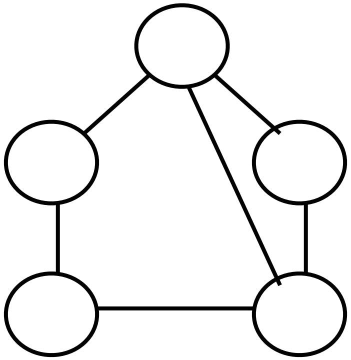
Figure 4.8: Directional path-consistency (DPC)

Theorem 4.3.7 *Given a binary network $\mathcal{R}$ and an ordering $\mathbf{d}$, algorithm DPC generates a largest equivalent, strong, directional-path-consistent network relative to $\mathbf{d}$. 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.*

Last slide in class

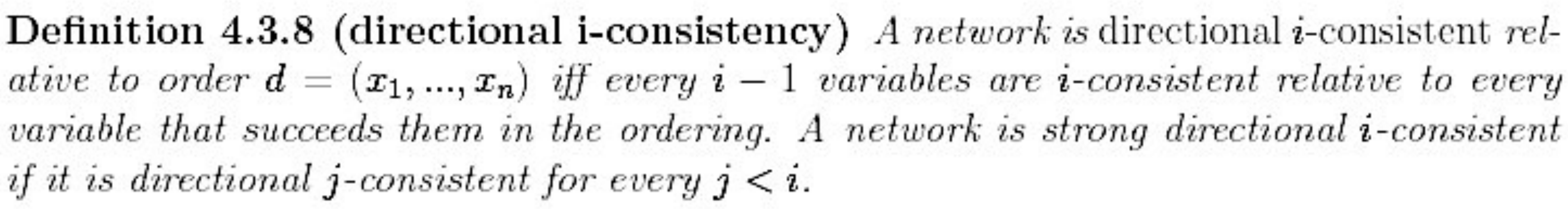
Example of DPC

- d=A,B,C,D,E



R_{CB} = { (1,3)(2,3)}
R_{DB} = {((1,1)(2,2)}
R_{CD}= {(1,1)(2,2)(1,3)(2,3)}

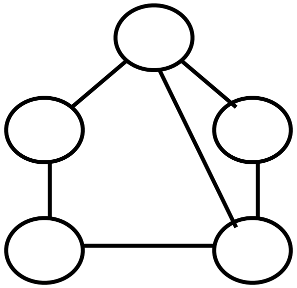
Directional i-consistency

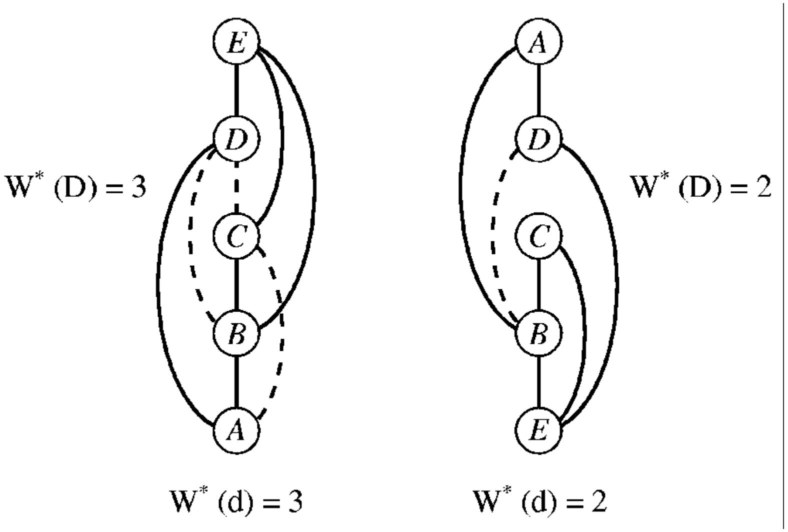


Definition 4.3.8 (directional i-consistency) *A network is* directional i-consistent *relative to order* $\mathbf{d} = (x_1, ..., x_n)$ *iff every* $i - 1$ *variables are* i*-consistent relative to every variable that succeeds them in the ordering. A network is strong directional* i*-consistent if it is directional* j*-consistent for every* $j < i$.

The induced-width

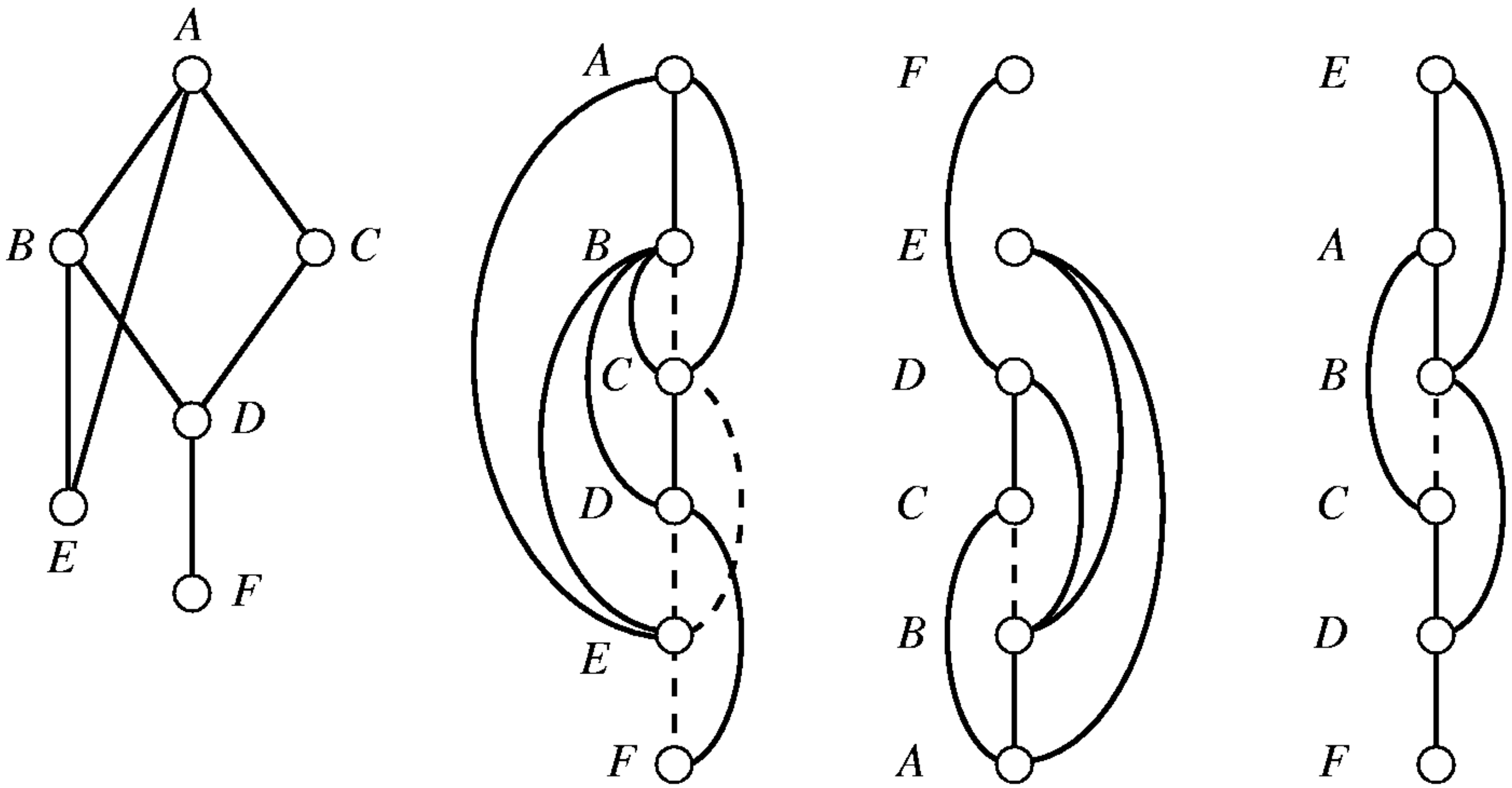
DPC recursively connects parents in the ordered graph, yielding
Induced-ordered graph:





- Width along ordering *d*, w(d):
  - max # of previous parents in the Original graph
- Induced width w*(d):
  - The width in the ordered *induced graph: defined by recursively connecting the parents from last to first*
- Induced-width w*:
  - Smallest induced-width over all orderings
- Finding w*
  - NP-complete *(Arnborg, 1985) but greedy heuristics (min-fill).*

Induced-width (continued)

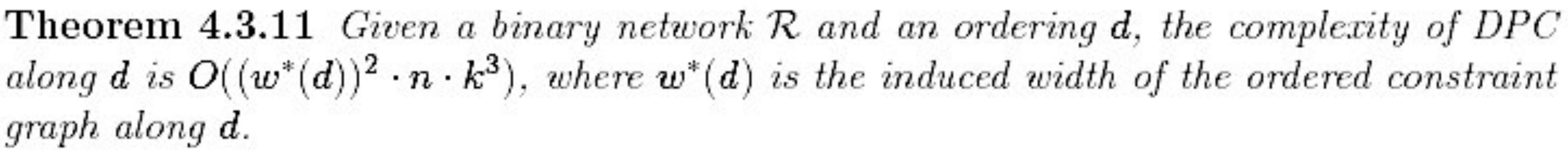


(a) (b) (c) (d)

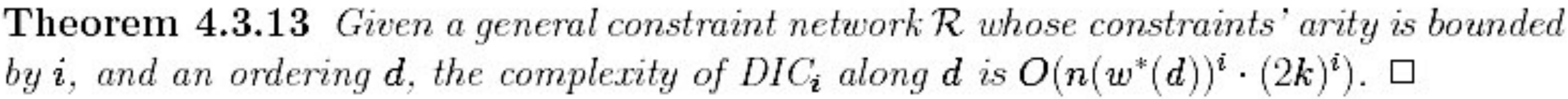
Induced-width and DPC

- 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 i-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* $w^*(d)$ *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* $O(n(w^*(d))^i \cdot (2k)^i)$. ☐

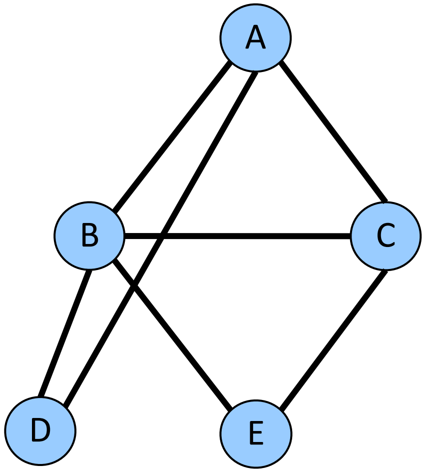
- Consequently we wish to have ordering with minimal induced-width
- Induced-width is equal to tree-width to be defined later.
- Finding min induced-width ordering is NP-complete

Outline

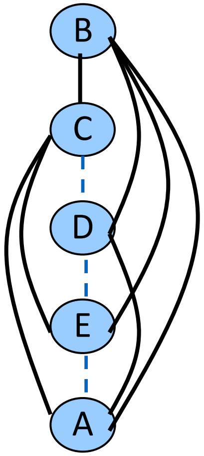
- Directional Arc-consistency algorithms
- Directional Path-consistency and directional i-consistency
- Greedy algorithms for induced-width
- Width and local consistency
- Adaptive-consistency and bucket-elimination

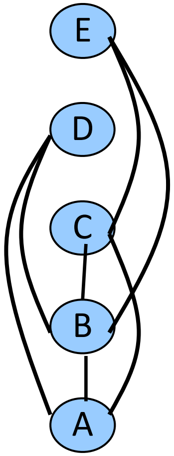
How to find a good induced-width greedily

The effect of the ordering:



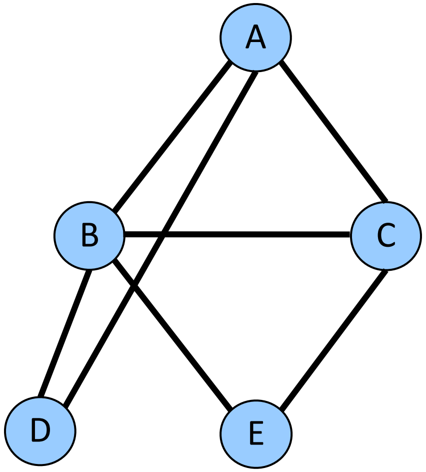
Primal (moraal) graph





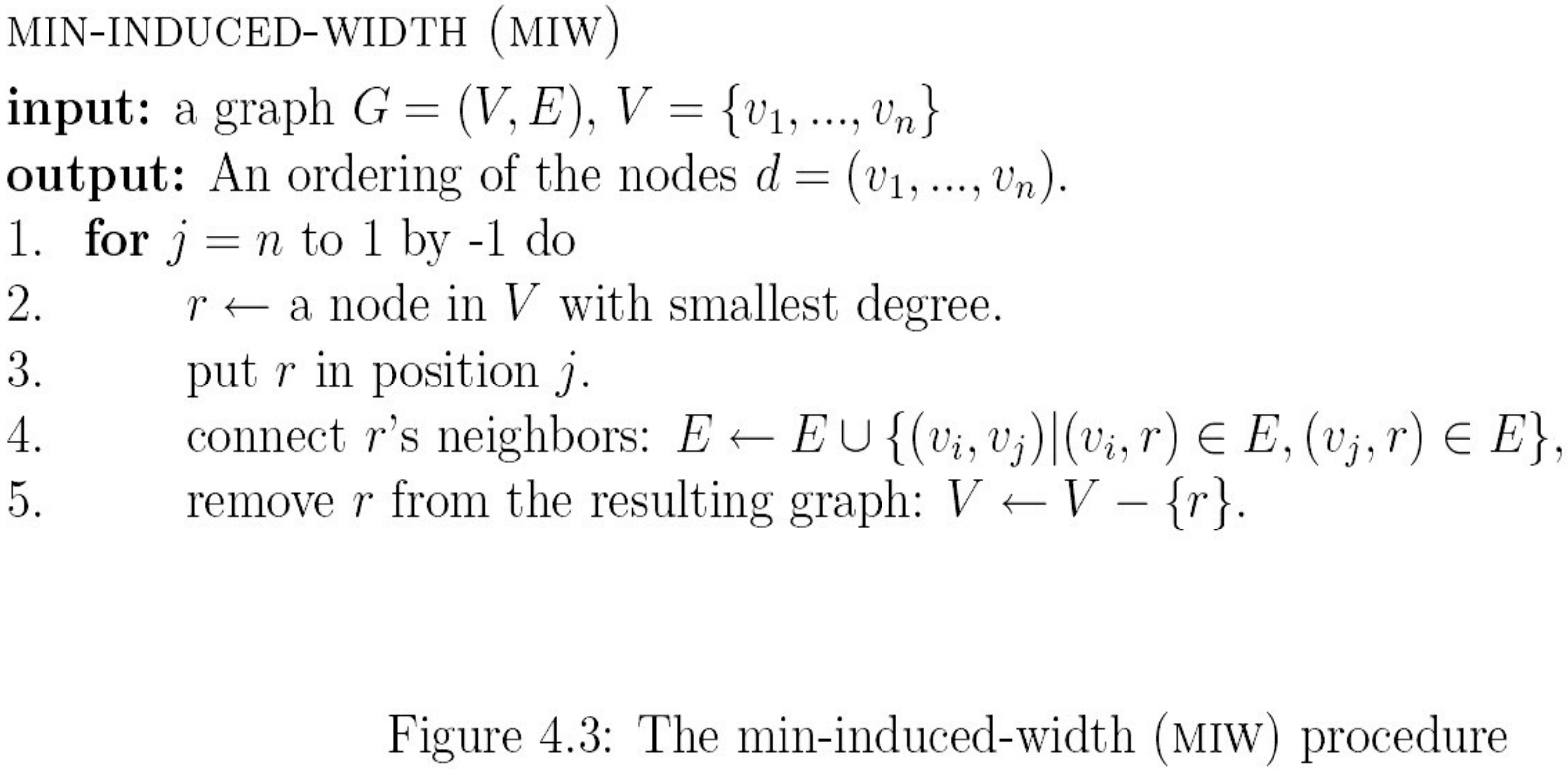
Greedy algorithms for induced-width

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



Primal (moraal) graph

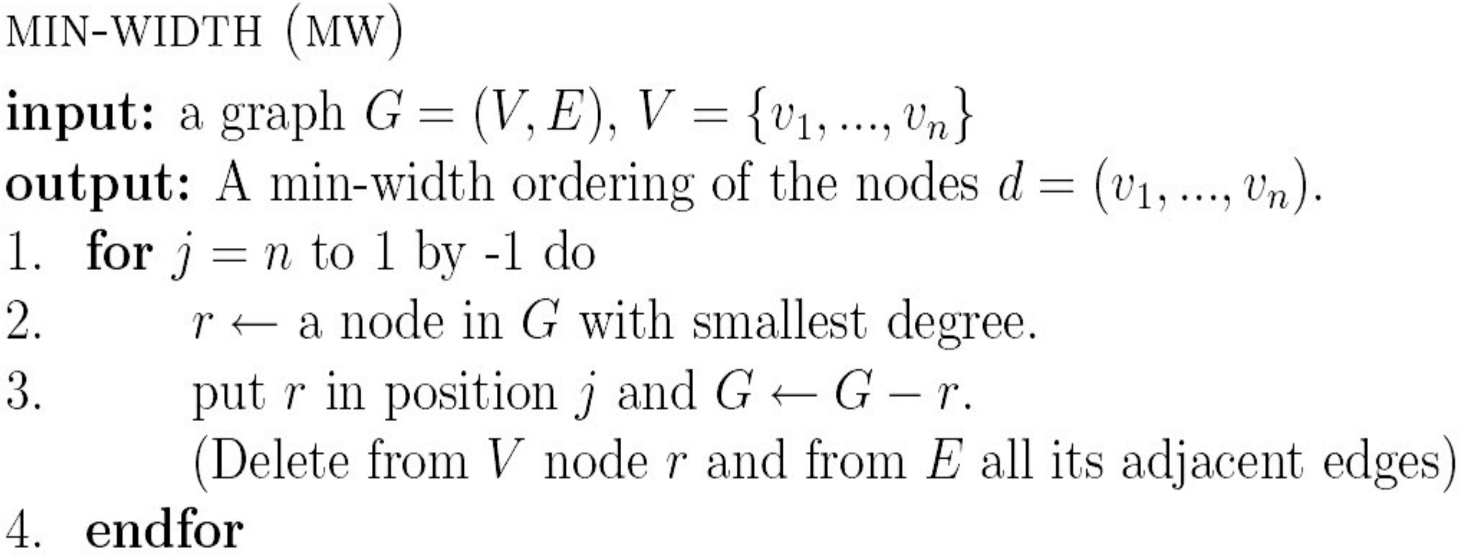
Min-induced-width



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

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

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 $d = (v_1, ..., v_n)$.
for $j = n$ to 1 by -1 do
        $r \leftarrow$ a node in $G$ with smallest degree.
        put $r$ in position $j$ and $G \leftarrow G - r$.
        (Delete from $V$ node $r$ and from $E$ all its adjacent edges)
endfor
```

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

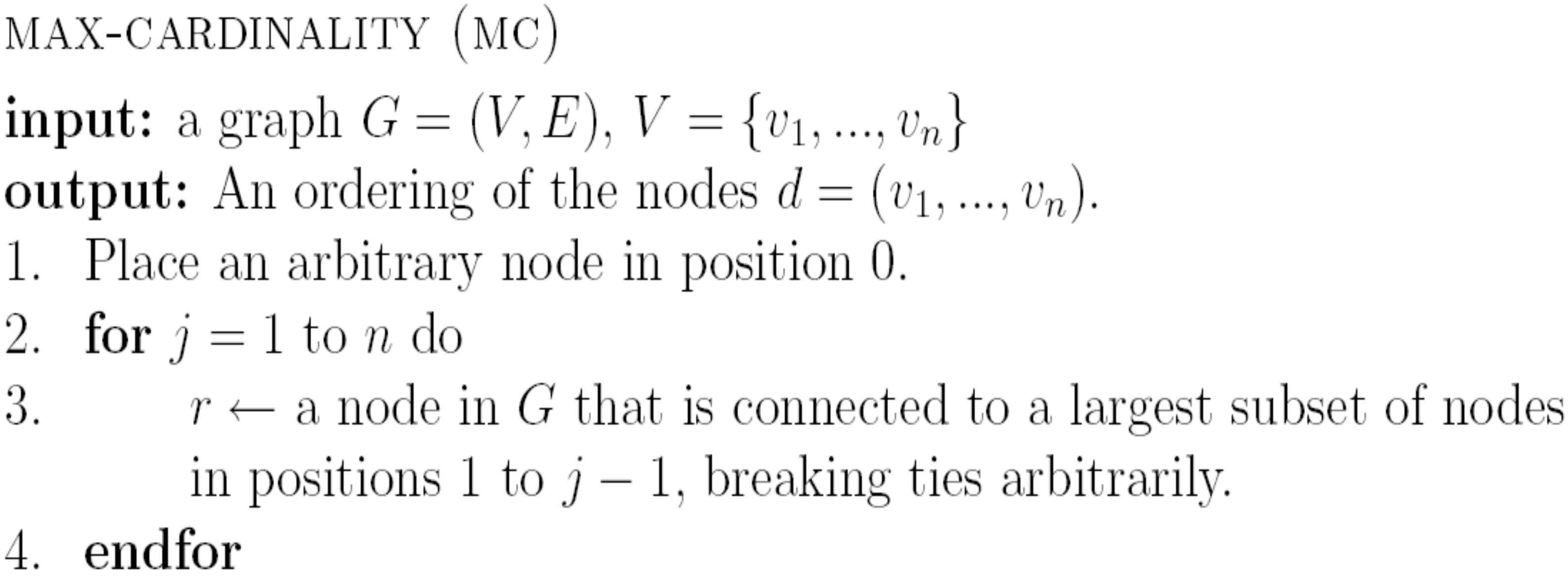
Min-fill algorithm

- Prefers a node who adds the least number of fill-in arcs.
- Empirically, fill-in is the best among the greedy algorithms (MW,MIW,MF,MC)

Chordal graphs and 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
- If G* is an induced graph it 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

Max-cardinality ordering

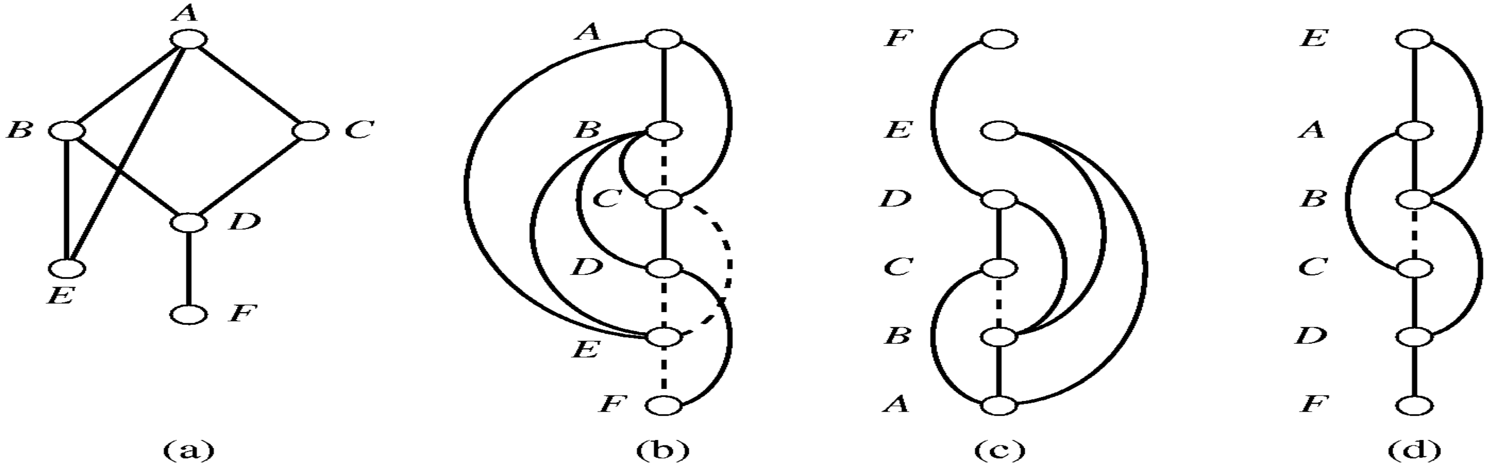


```
MAX-CARDINALITY (MC)
input: a graph $G = (V, E)$, $V = \{v_1, ..., v_n\}$
output: An ordering of the nodes $d = (v_1, ..., v_n)$.
Place an arbitrary node in position 0.
for $j = 1$ to $n$ do
        $r \leftarrow$ a node in $G$ that is connected to a largest subset of nodes
        in positions 1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

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

Example

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.



Outline

- Directional Arc-consistency algorithms
- Directional Path-consistency and directional i-consistency
- Greedy algorithms for induced-width
- Width and local consistency
- Adaptive-consistency and bucket-elimination

Width vs local consistency: solving trees

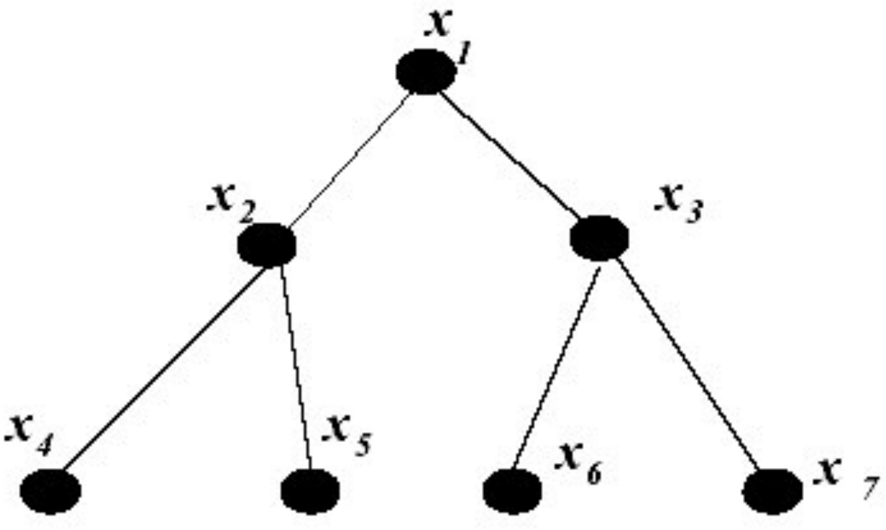
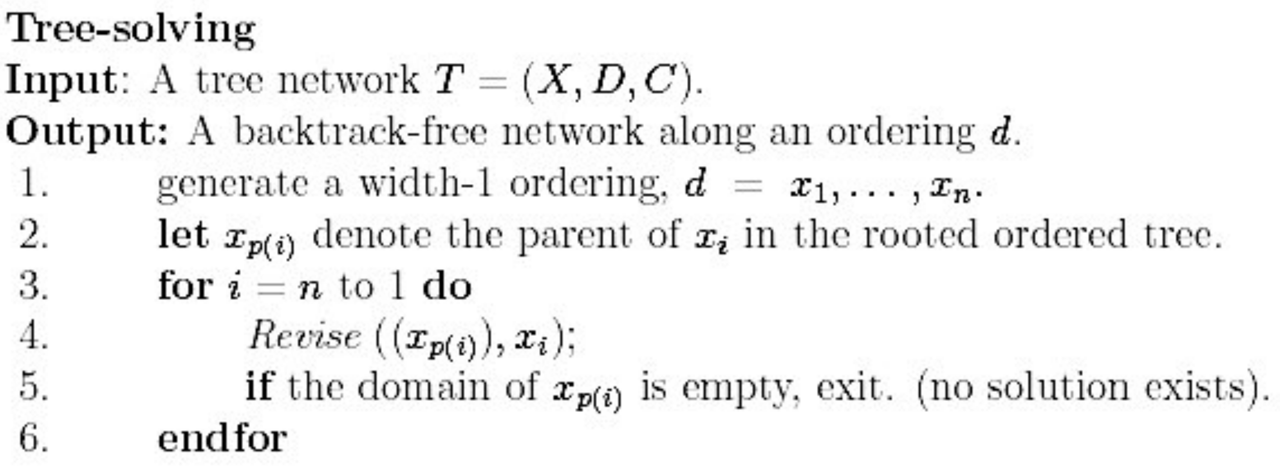


Figure 4.10: A tree network

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

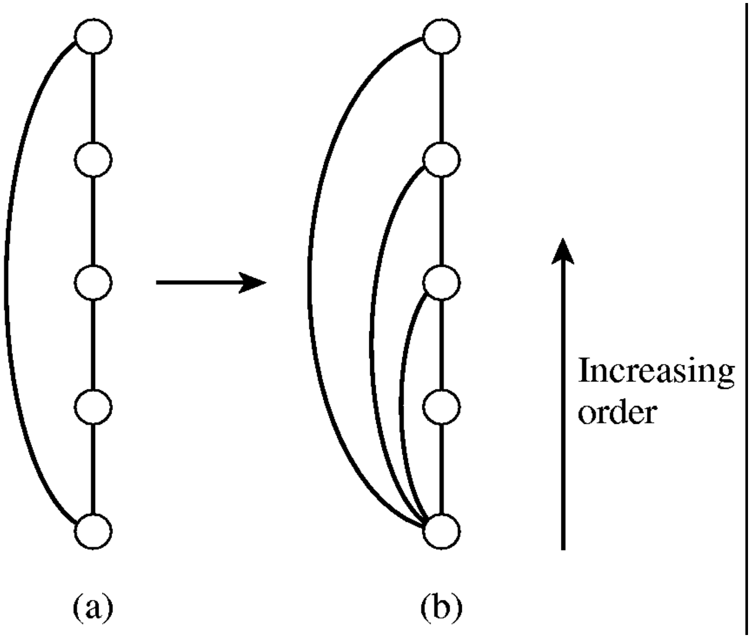
Input: A tree network $T = (X, D, C)$.

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

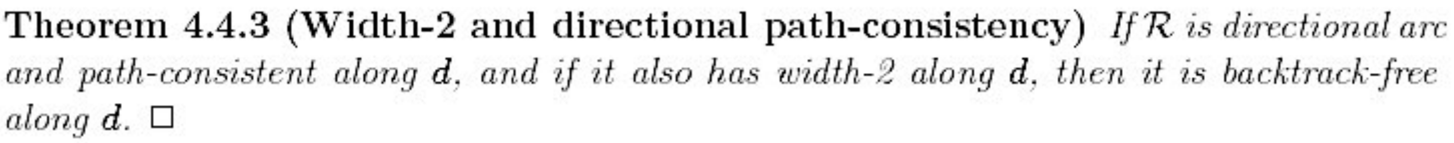
```
generate a width-1 ordering, $d = x_1, \ldots, x_n$.
let $x_{p(i)}$ denote the parent of $x_i$ in the rooted ordered tree.
for $i = n$ to 1 do
    Revise $((x_{p(i)}), x_i)$;
    if the domain of $x_{p(i)}$ is empty, exit. (no solution exists).
endfor
```

Figure 4.11: Tree-solving algorithm

Width-2 and DPC



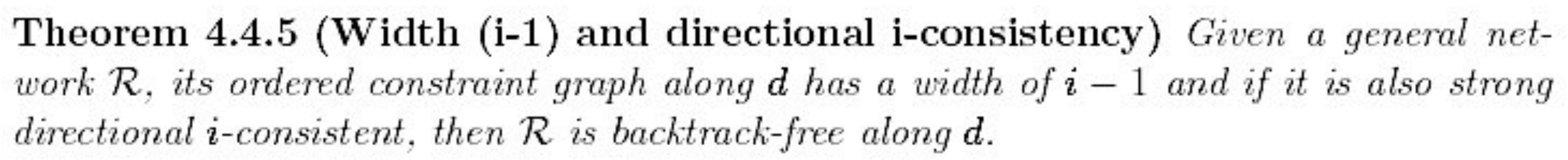
(a) (b)



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)



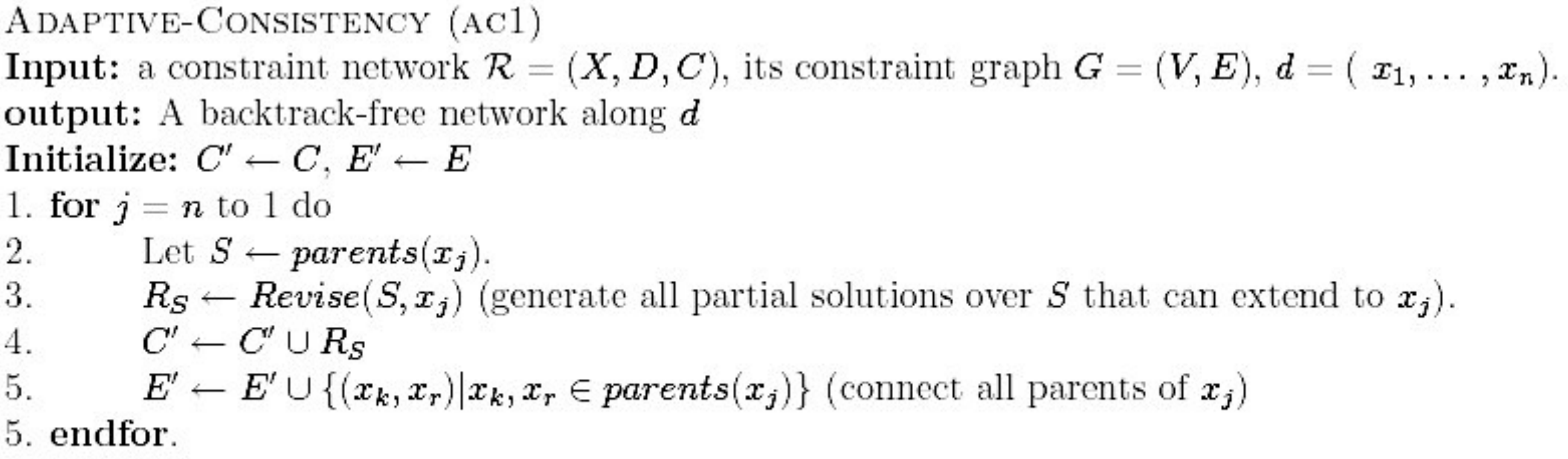
Theorem 4.4.5 (Width (i-1) and directional i-consistency) *Given a general network $\mathcal{R}$, its ordered constraint graph along* $\mathbf{d}$ *has a width of* $\mathbf{i} - 1$ *and if it is also strong directional* $\mathbf{i}$*-consistent, then* $\mathcal{R}$ *is backtrack-free along* $\mathbf{d}$.

Width vs i-consistency

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

- If a problem has width 2, will DPC make it backtrack-free?
- **Adaptive-consistency**: applies i-consistency when i is adapted to the number of parents

Adaptive-consistency

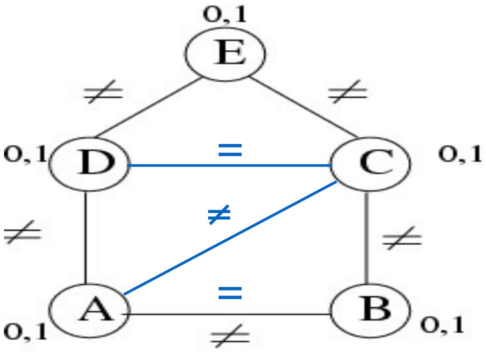


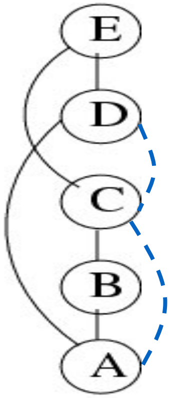
```
ADAPTIVE-CONSISTENCY (AC1)
Input: a constraint network $\mathcal{R} = (X, D, C)$, its constraint graph $G = (V, E)$, $d = (x_1, \ldots, x_n)$.
output: A backtrack-free network along $d$
Initialize: $C' \leftarrow C$, $E' \leftarrow E$
for $j = n$ to 1 do
    Let $S \leftarrow parents(x_j)$.
    $R_S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

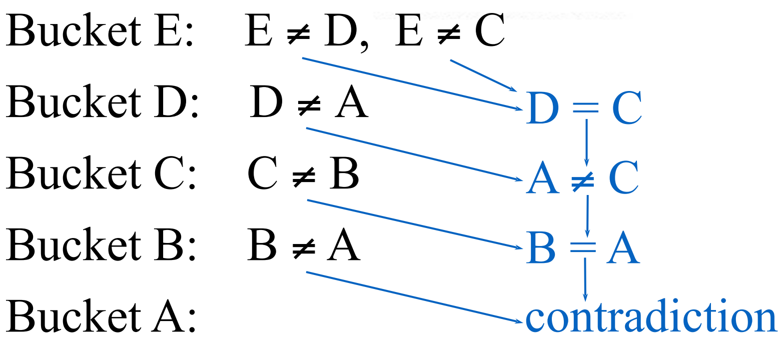
Figure 4.13: Algorithm adaptive-consistency– version 1

Bucket Elimination

Adaptive Consistency (Dechter & Pearl, 1987)







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

Bucket D: $D \neq A$ $D = C$

Bucket C: $C \neq B$ $A \neq C$

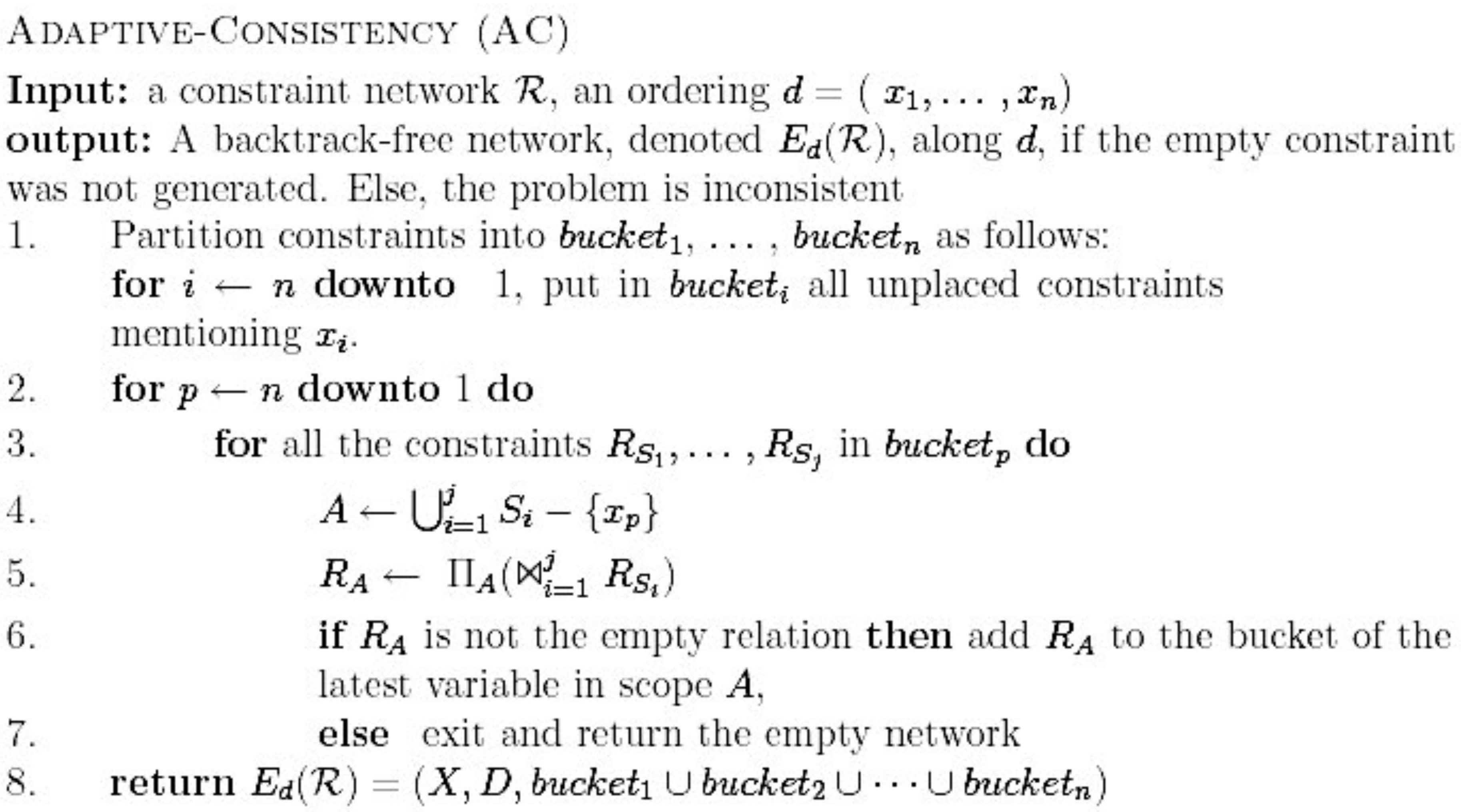
Bucket B: $B \neq A$ $B = A$

Bucket A: contradiction

Complexity: nk^{w^*+1}

w^* is the induced-width along the ordering

Adaptive-consistency, bucket-elimination



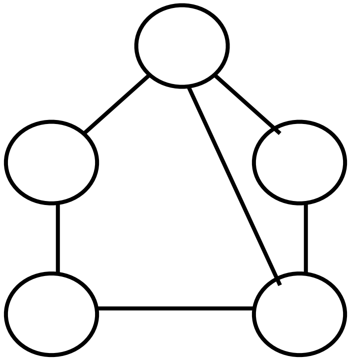
ADAPTIVE-CONSISTENCY (AC)

```
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
    Partition constraints into $bucket_1, \ldots , bucket_n$ as follows:
    for $i \leftarrow n$ downto 1, put in $bucket_i$ all unplaced constraints
    mentioning $x_i$.
    for $p \leftarrow n$ downto 1 do
        for all the constraints $R_{S_1}, \ldots , R_{S_j}$ in $bucket_p$ do
            $A \leftarrow \bigcup_{i=1}^{j} S_i - \{x_p\}$
            $R_A \leftarrow \Pi_A(\bowtie_{i=1}^{j} R_{S_i})$
            if $R_A$ is not the empty relation then add $R_A$ to the bucket of the
            latest variable in scope $A$,
            else exit and return the empty network
    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 & Pearl, 1987)

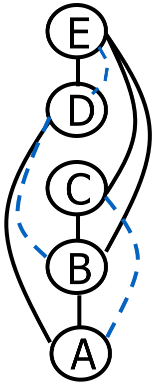


// R_{DCB}

// R_{ACB}

// R_{AB}

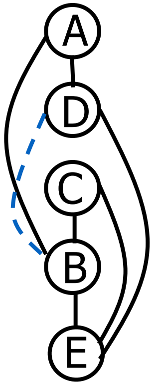
R_A



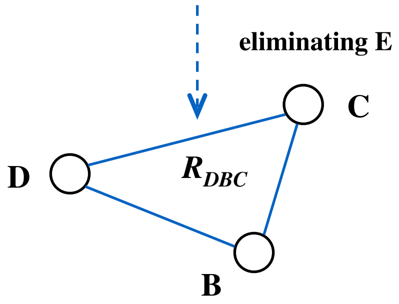
// R_{DB}

// R^{D}_{BE}, R^{C}_{BE}

// R_E



The Idea of Elimination

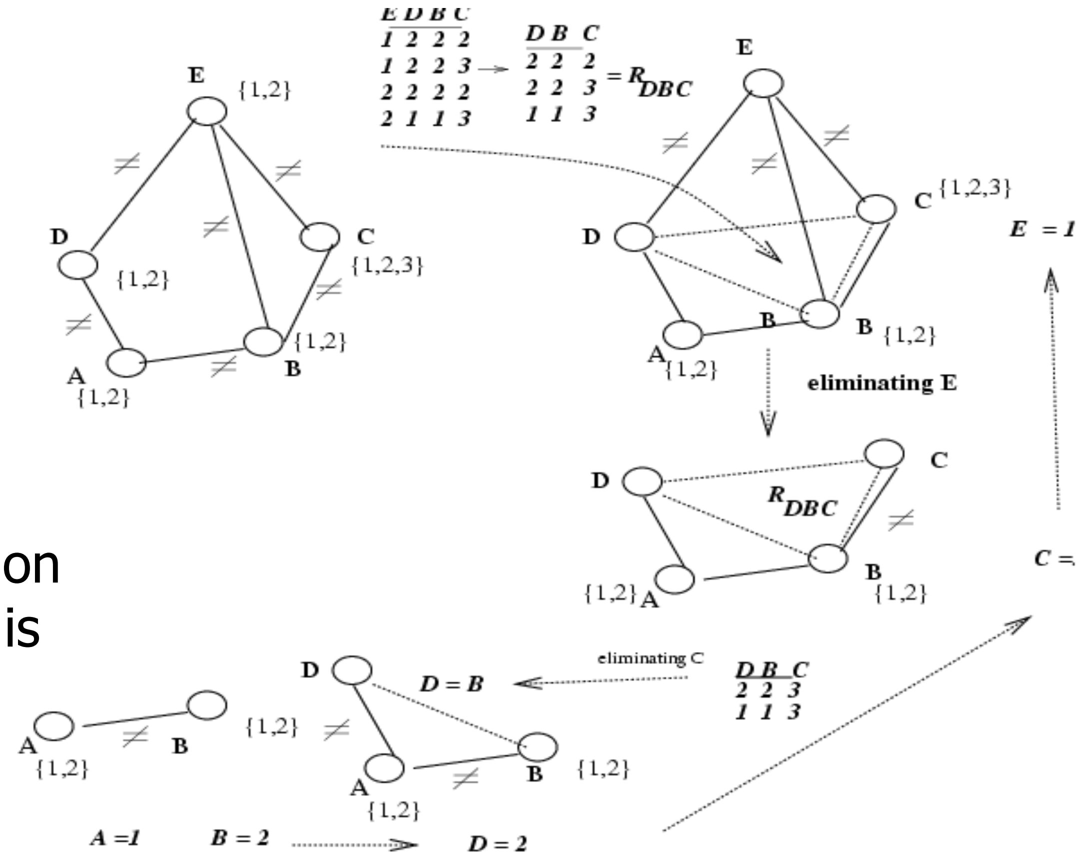


3
value assignment

$\bowtie$ $\bowtie$

Variable Elimination

Eliminate variables one by one: “constraint propagation”



Solution generation after elimination is backtrack-free

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 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).**

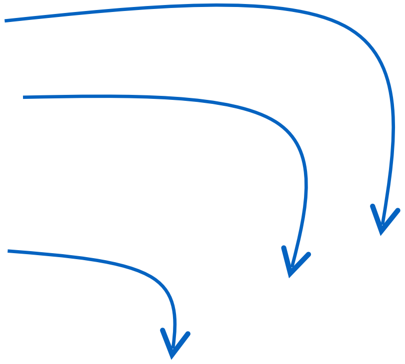
Back to Induced width

- 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.

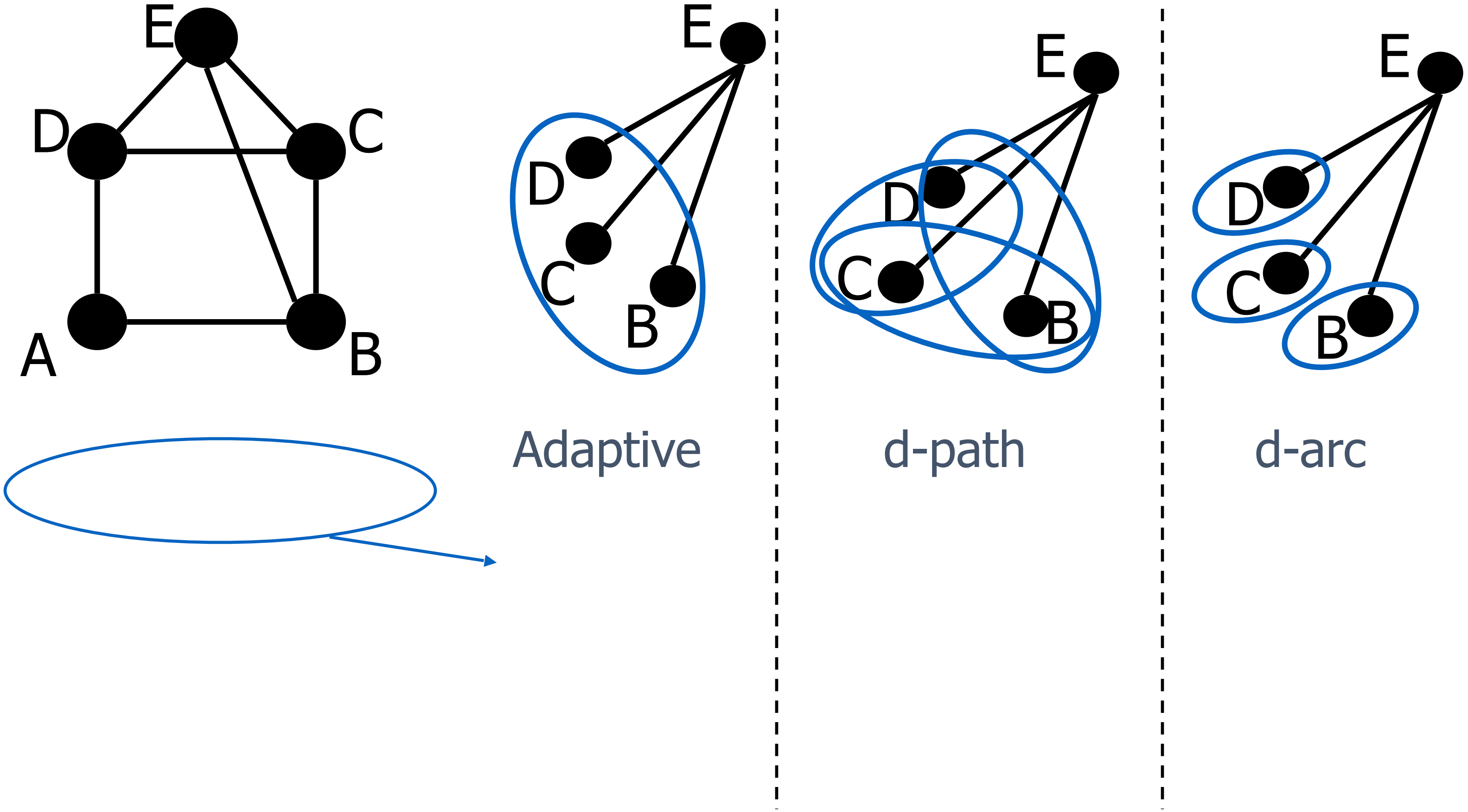
Solving Trees

(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



Relational consistency
(Chapter 8)

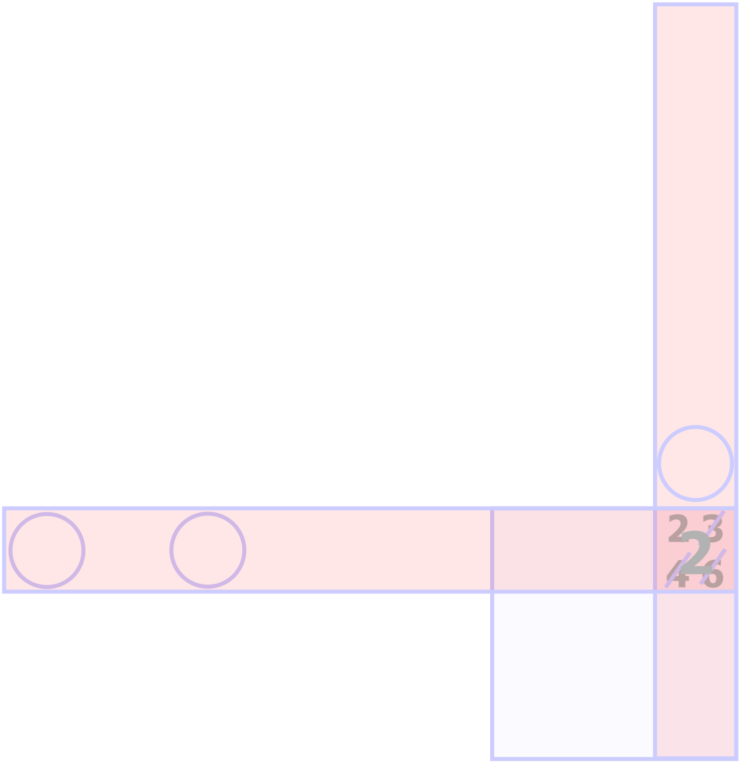
- Relational arc-consistency
- Relational path-consistency
- Relational m-consistency
- Relational consistency for Boolean and linear constraints:
  - Unit-resolution is relational-arc-consistency
  - Pair-wise resolution is relational path-consistency

Sudoku’s propagation

- http://www.websudoku.com/
- What kind of propagation we do?

Sudoku

Constraint propagation

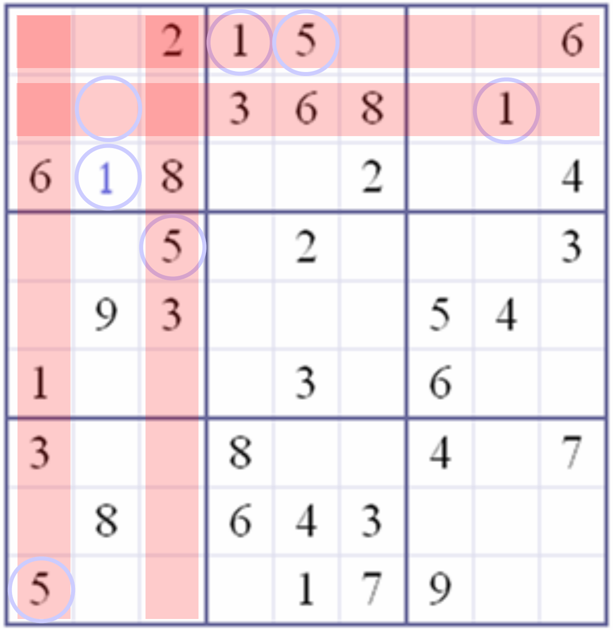


- **Variables: 81 slots**
- **Domains = {1,2,3,4,5,6,7,8,9}**
- **Constraints:**
  - **27 not-equal**

Each row, column and major block must be alldifferent

“Well posed” if it has unique solution: 27 constraints

Sudoku



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

Each row, column and major block must be alldifferent

“Well posed” if it has unique solution