# CompSci 275, CONSTRAINT Networks

Rina Dechter, Fall 2020

## Directional consistency
## Chapter 4

# Outline

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

# 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

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

# Boolean constraint propagation

**Procedure** UNIT-PROPAGATION
**Input:** A cnf theory, $\varphi$, $d = Q_1, ..., Q_n$.
**Output:** An equivalent theory such that every unit clause does not appear in any non-unit clause.

```
queue = all unit clauses.
while queue is not empty, do.
        T ← next unit clause from Queue.
        for every clause β containing T or ¬T
                if β contains T delete β (subsumption elimination)
                else, For each clause γ = resolve(β, T).
                if γ, the resolvent, is empty, the theory is unsatisfiable.
                else, add the resolvent γ to the theory and delete β.
                if γ is a unit clause, add to Queue.
        endfor.
endwhile.
```

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

# Consistency for numeric constraints (Gausian elimination)

$x \in [1,10], y \in [5,15],$

$x + y = 10$

$arc - consistency \Rightarrow x \in [1,5], y \in [5,9]$

Gausian elimination of $x + y = 10, -y \leq -5$

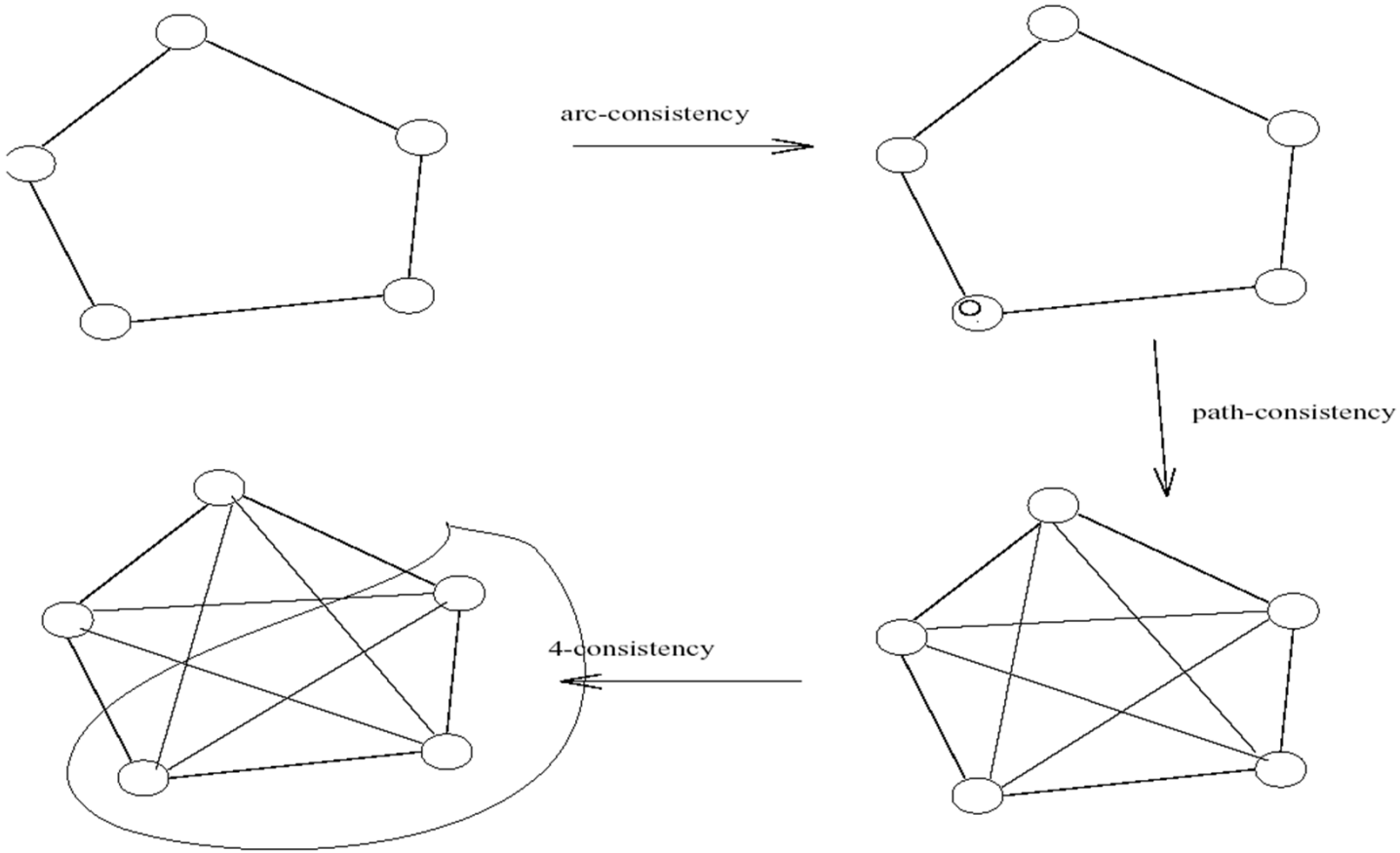
$z \in [-10,10],$

$y + z \leq 3$

$path - consistency \Rightarrow x - z \geq 7$

Gausian Elinination of: $, x + y = 10, -y - z \geq -3$

# Impact on graphs of i-consistency



# Tractable classes

**Theorem 3.7.1** *1. The consistency binary constraint networks having no cycles can be decided by arc-consistent*

*2. The consistency of binary constraint networks with bi-valued domains can be decided by path-consistency,*

*3. The consistency of Horn cnf theories can be decided by unit propagation.*

- Examples of Horn theories (each clause has at most one positive literal)
- (¬ A, ¬ B , ¬ C, D), (¬ D, F), (¬ A)

# 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

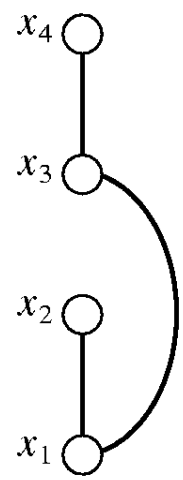
Let's explore how we can make a problem backtrack-free with a minimal amount of effort

**Definition 4.1.1 (backtrack-free search)** *A constraint network is backtrack-free relative to a given ordering* $d = (x_1, ..., x_n)$ *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

**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* $i \leq j$.




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

# 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.

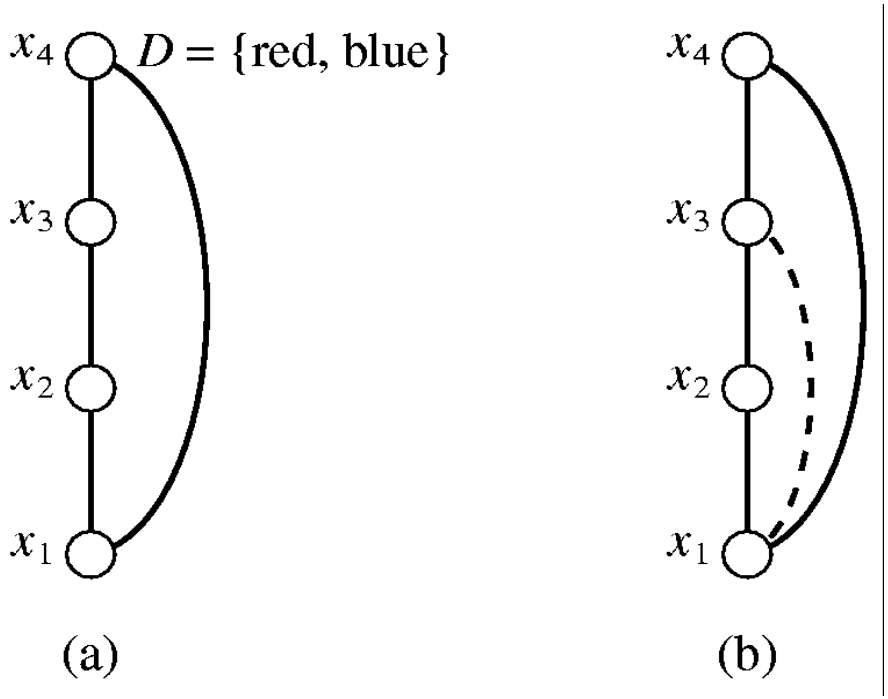
1. **for** $i = n$ to 1 by $-1$ **do**
2. **for** each $j < i$ s.t. $R_{ji} \in \mathcal{R}$,
3. $D_j \leftarrow D_j \cap \pi_j(R_{ji} \bowtie D_i)$, (this is revise$((x_j), x_i)$).
4. **end-for**

Figure 4.6: Directional arc-consistency (DAC)

- Complexity: $O(ek^2)$

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

Not equal constraints
Is it arc-consistent?




**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:** $E' \leftarrow E$.

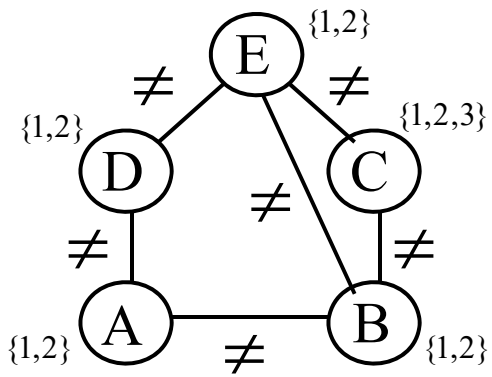
1. **for** $k = n$ to 1 by -1 **do**
2. (a) $\forall\ i \leq k$ such that $x_i$ is connected to $x_k$ in the graph, **do**
3. $D_i \leftarrow D_i \cap \pi_i(R_{ik} \bowtie D_k)$ $(Revise((x_i), x_k))$
4. (b) $\forall\ i, j \leq k$ s.t. $(x_i, x_k)$,$(x_j, x_k) \in E'$ **do**
5. $R_{ij} \leftarrow R_{ij}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

Figure 4.8: Directional path-consistency (DPC)

**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.*

# Example of DPC

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



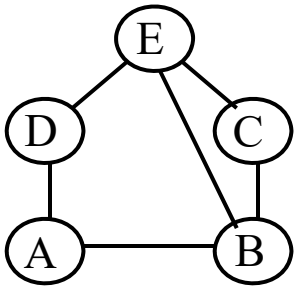

R_CB = { (1,3)(2,3)}
R_DB = {((1,1)(2,2)}
R_DC = {(1,1)(2,2)(1,3)(2,3)}

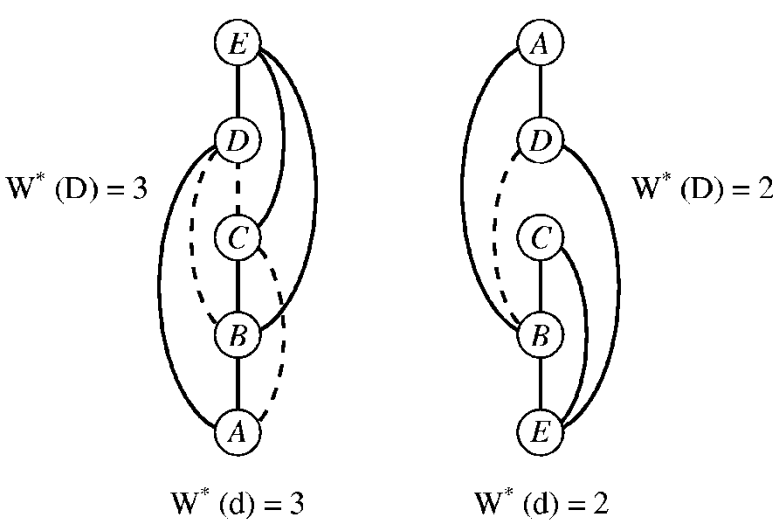
# Directional i-consistency

**Definition 4.3.8 (directional i-consistency)** *A network is* directional $i$-consistent *relative to order* $d = (x_1, ..., x_n)$ *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

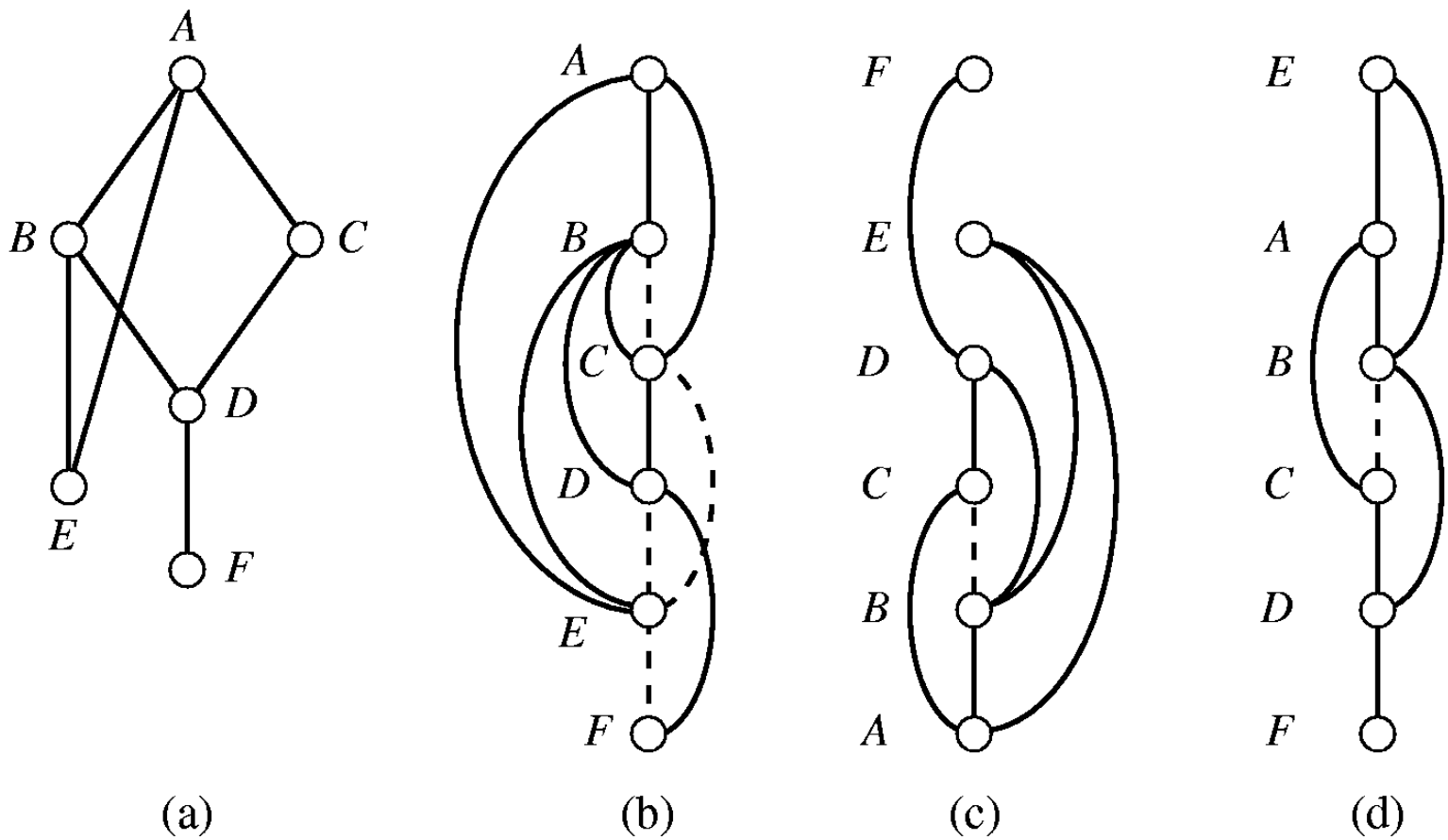
DPC recursively connects parents in the ordered graph, yielding Induced-ordered graph:




- Width along ordering *d*, w(d):
  - max # of previous parents
- Induced width w*(d):
  - The width in the ordered *induced graph: 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)



# 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 $d$, the complexity of DPC along $d$ is $O((w^*(d))^2 \cdot n \cdot k^3)$, where $w^*(d)$ is the induced width of the ordered constraint graph along $d$.*

**Theorem 4.3.13** *Given a general constraint network $\mathcal{R}$ whose constraints' arity is bounded by $i$, and an ordering $d$, the complexity of $DIC_i$ along $d$ 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

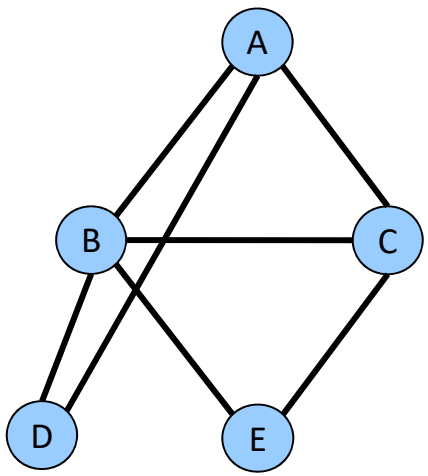
# Outline

- 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

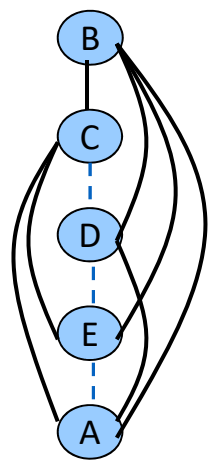
# How to find a good induced-width greedily

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

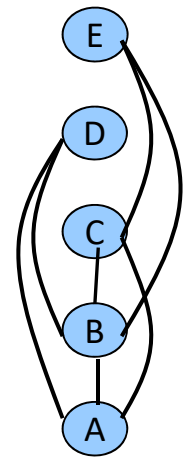
The effect of the ordering:




Primal (moraal) graph



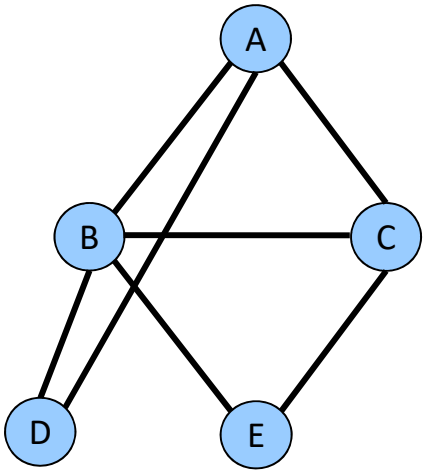

$w^*(d_1) = 4$




$w^*(d_2) = 2$

# 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 $d = (v_1, ..., v_n)$.

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-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-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 $G = (V, E)$, $V = \{v_1, ..., v_n\}$

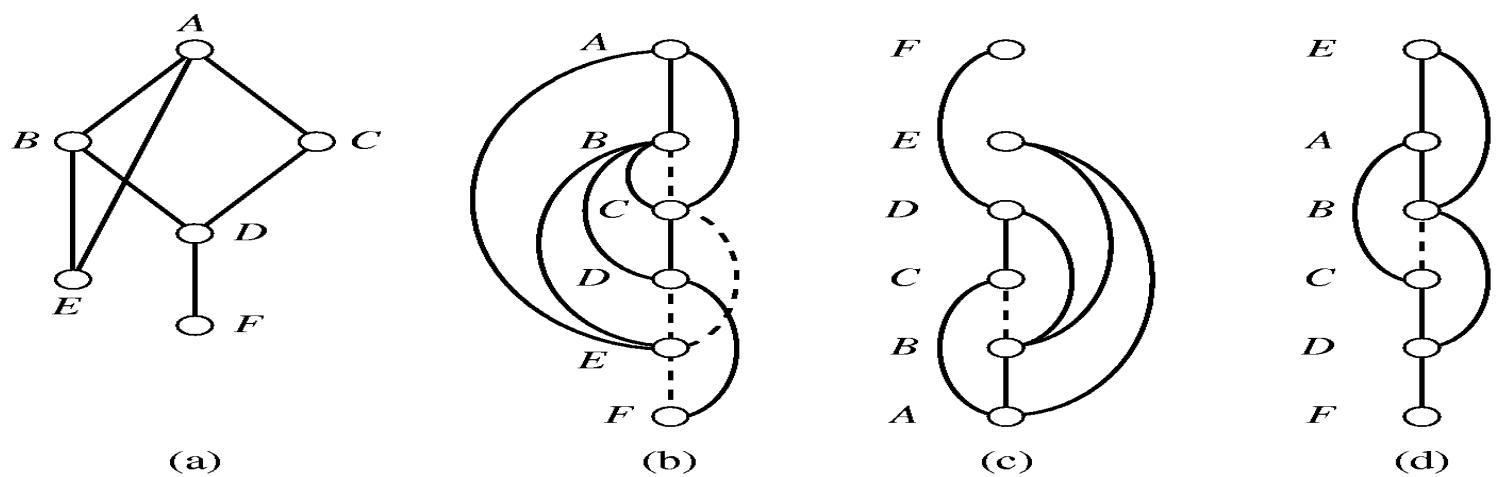
**output:** An ordering of the nodes $d = (v_1, ..., v_n)$.

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**

**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

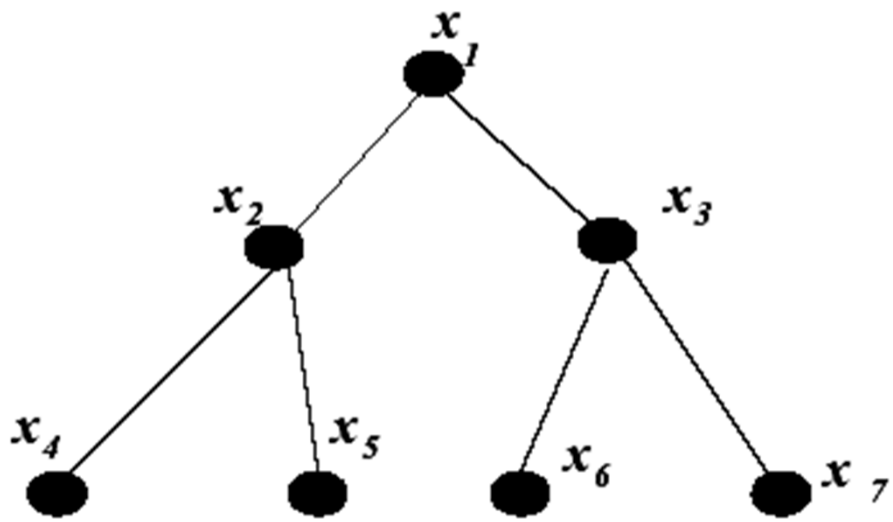



Figure 4.10: A tree network

**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

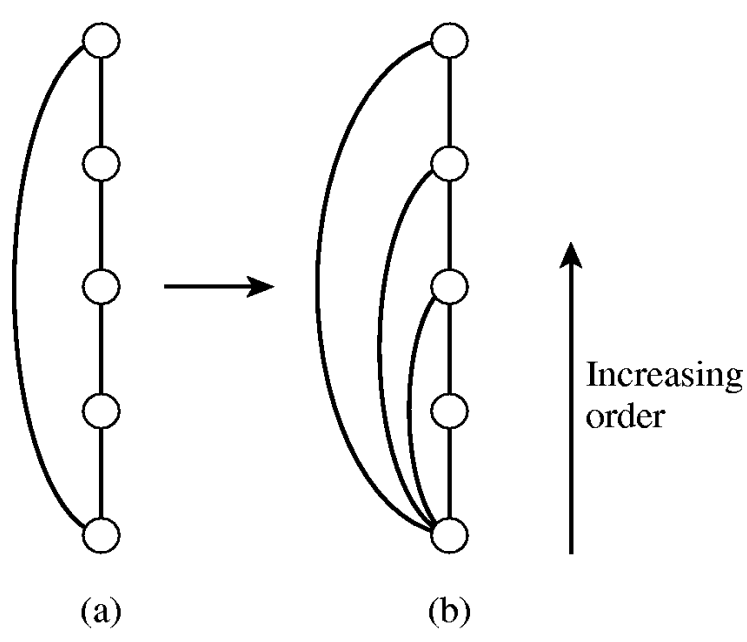
**Tree-solving**
**Input**: A tree network $T = (X, D, C)$.
**Output:** A backtrack-free network along an ordering $d$.

```
generate a width-1 ordering, d = x_1, ..., 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

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

# Width-2 and DPC




**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 $i$-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

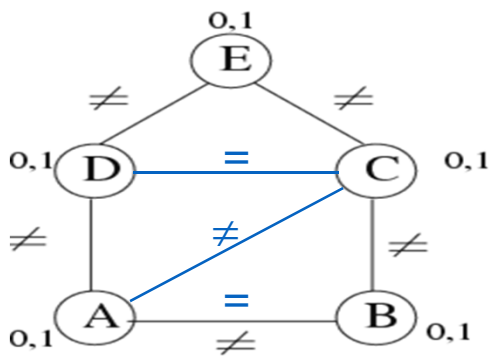
# Adaptive-consistency

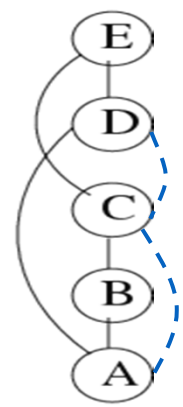
ADAPTIVE-CONSISTENCY (AC1)
**Input:** a constraint network $\mathcal{R} = (X, D, C)$, its constraint graph $G = (V, E)$, $d = (x_1, \ldots, x_n)$.
**output:** A backtrack-free network along $d$
**Initialize:** $C' \leftarrow C$, $E' \leftarrow E$
1. **for** $j = n$ to 1 **do**
2. Let $S \leftarrow parents(x_j)$.
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

# 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 $d = (x_1, \ldots, x_n)$

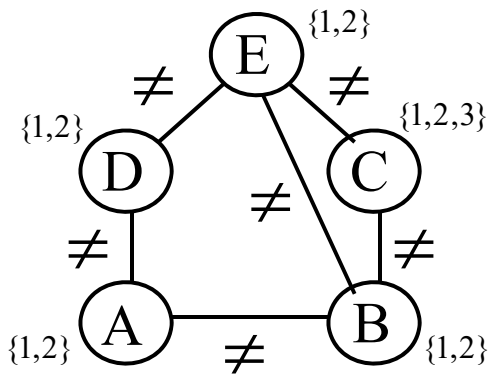
**output:** A backtrack-free network, denoted $E_d(\mathcal{R})$, along $d$, if the empty constraint was not generated. Else, the problem is inconsistent

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 & Pearl, 1987)



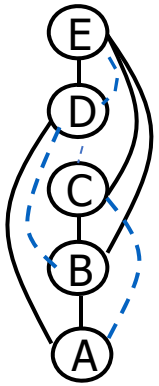

$Bucket(E)$: $E \neq D,\ E \neq C,\ E \neq B$

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

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

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

$Bucket(A)$: $R_A$



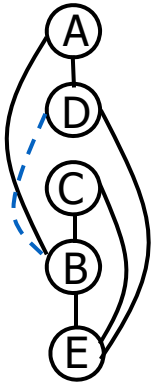

$Bucket(A)$: $A \neq D,\ A \neq B$

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

$Bucket(C)$: $C \neq B,\ C \neq E$

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

$Bucket(E)$: $\|\ R_E$

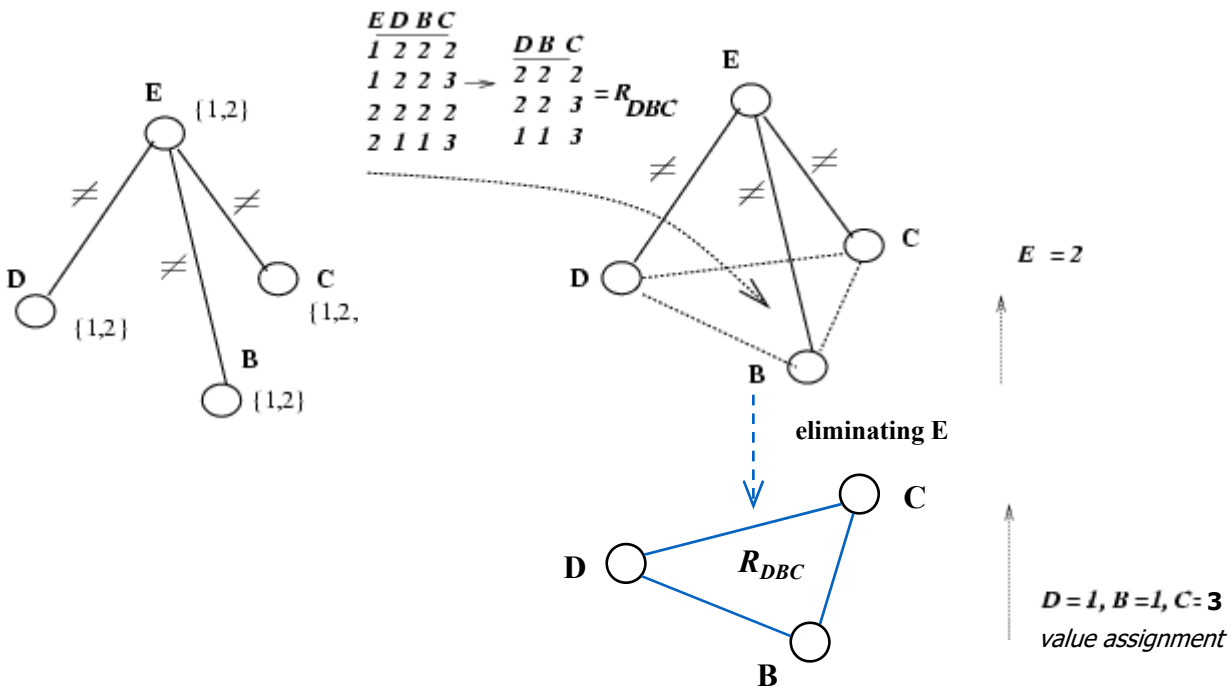



**Time :** $O(n \exp(w^*(d)+1))$,

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

$w^*(d)$ - ***induced - width - along - ordering - d***

# The Idea of elimination



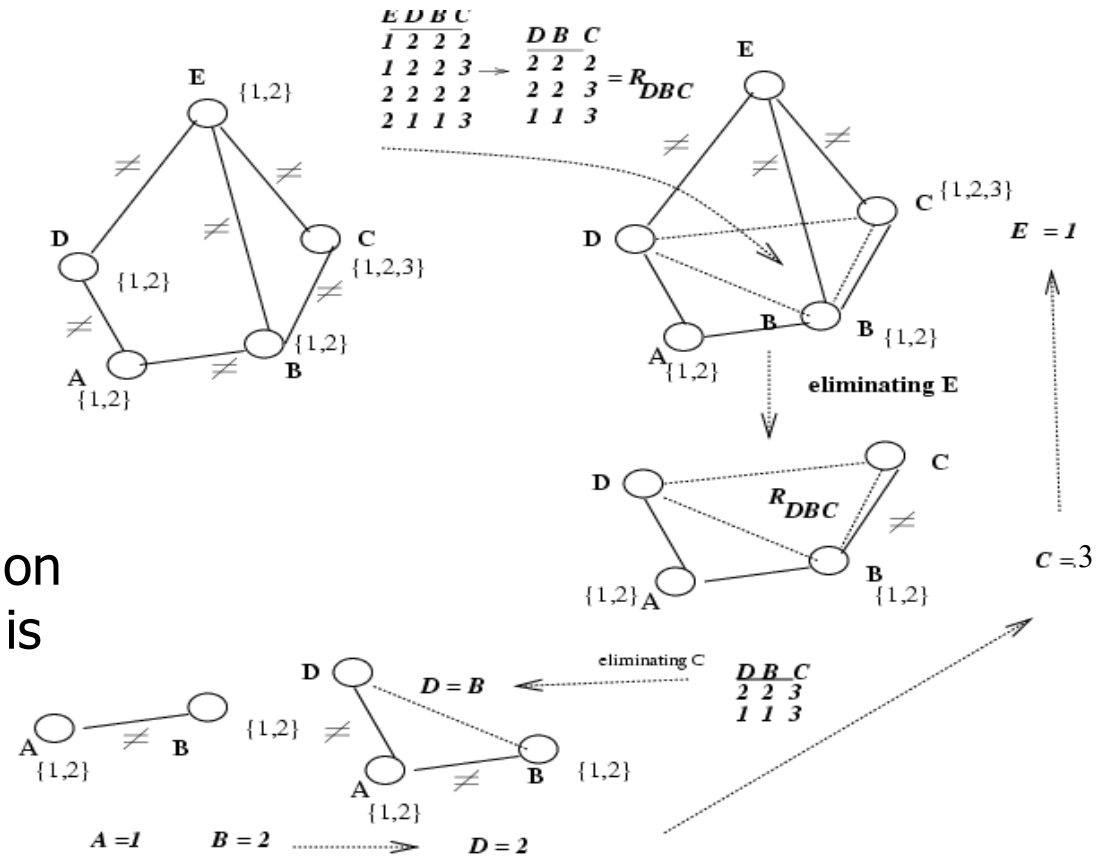

$$R_{DBC} = \prod_{DBC} R_{ED} \bowtie R_{EB} \bowtie R_{EC}$$

Eliminate variable E $\Leftrightarrow$ join and project

# Variable elimination

Eliminate variables one by one: "constraint propagation"

Solution generation after elimination is backtrack-free

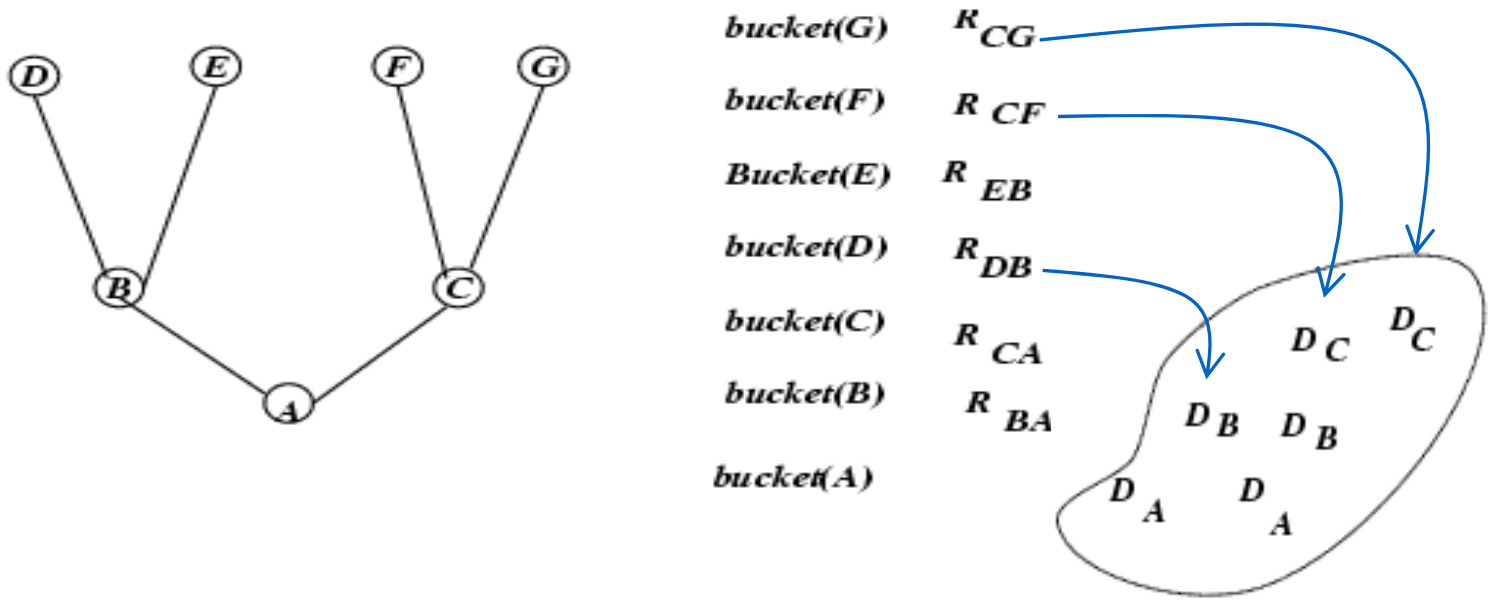


# Back to Induced width

- 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)



# 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

# 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

# Sudoku's propagation

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

# Sudoku

**Constraint propagation**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 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 |
| | | 9 | | | 4 | 5 | 8 | 1 |
| | | | 3 | | 2 | 9 | | |

- **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**