Constraint Satisfaction Problems

CSEN 266

Artificial Intelligence

Map Coloring Problem

- Look at the map of Australia
- Seven regions:
- Task: assign each region a color
  - Red, green, or blue
  - No neighboring regions have the same color

What is a CSP?

- Formulate the problem as a CSP
  - **Variables:** the regions X={WA, NT, Q, NSW, V, SA, T}
  - The **domain** of each variable is the set D_i={red, green, blue}
  - **Constraints:** neighboring regions have distinct colors

  9 places where regions boarder: 9 constraints

C={SA≠WA, SA ≠NT, SA ≠Q, SA ≠NSW, SA ≠V, WA ≠ NT, NT ≠ Q, Q ≠ NSW, NSW ≠V}

Defining a CSP

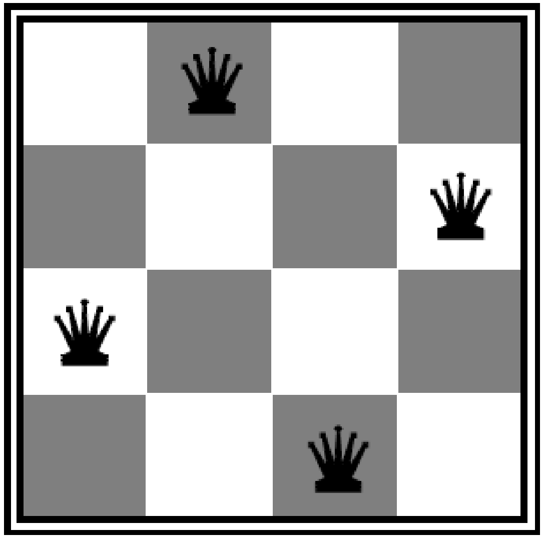
- Three components
  - X: a set of variables $\{X_1, X_2, \cdots, X_n\}$
  - D: a set of domains $\{D_1, D_2, \cdots, D_n\}$, one for each variable
    - Each domain D_i: a set of allowable values, $\{v_1, \cdots, v_k\}$ for variable X_i
  - C: a set of constraints that specify allowable combination of values for the variables

The Goal of a CSP

- Find an assignment of the values for the variables, such that the constraints are not violated.

Example: N-Queens

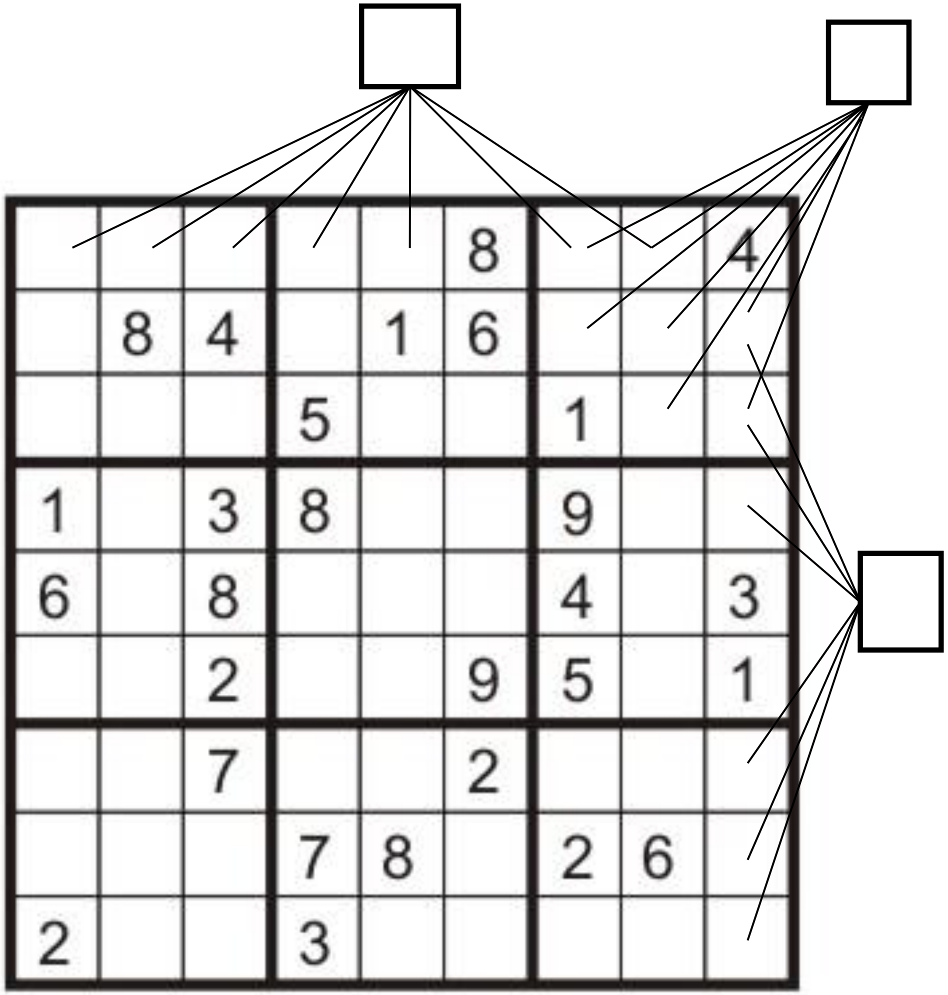
- Variables: X_{ij}
- Domains: {0, 1}
- Constraints:



$$\forall i, j, k \;\; (X_{ij}, X_{ik}) \in \{(0,0),(0,1),(1,0)\}$$
$$\forall i, j, k \;\; (X_{ij}, X_{kj}) \in \{(0,0),(0,1),(1,0)\}$$
$$\forall i, j, k \;\; (X_{ij}, X_{i+k,j+k}) \in \{(0,0),(0,1),(1,0)\}$$
$$\forall i, j, k \;\; (X_{ij}, X_{i+k,j-k}) \in \{(0,0),(0,1),(1,0)\}$$

$$\sum_{i,j} X_{ij} = N$$

Example: Sudoku



- Variables:
  - Each (open) square
- Domains:
  - {1,2,…,9}
- Constraints:

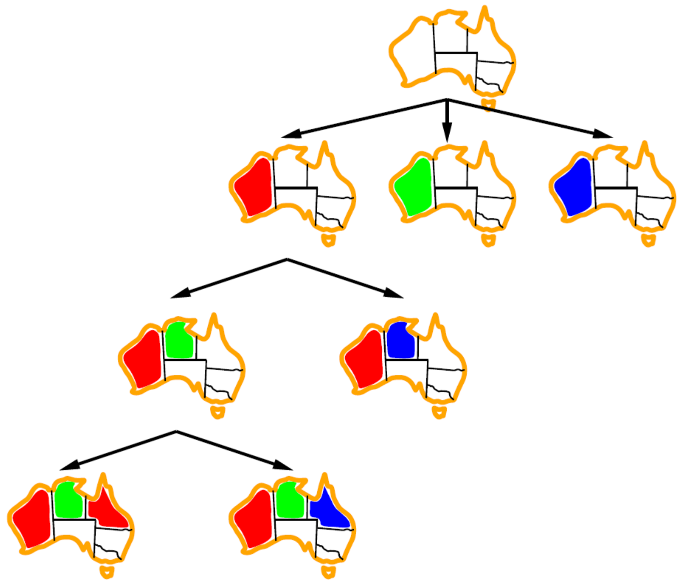
9-way alldiff for each column

9-way alldiff for each row

9-way alldiff for each region

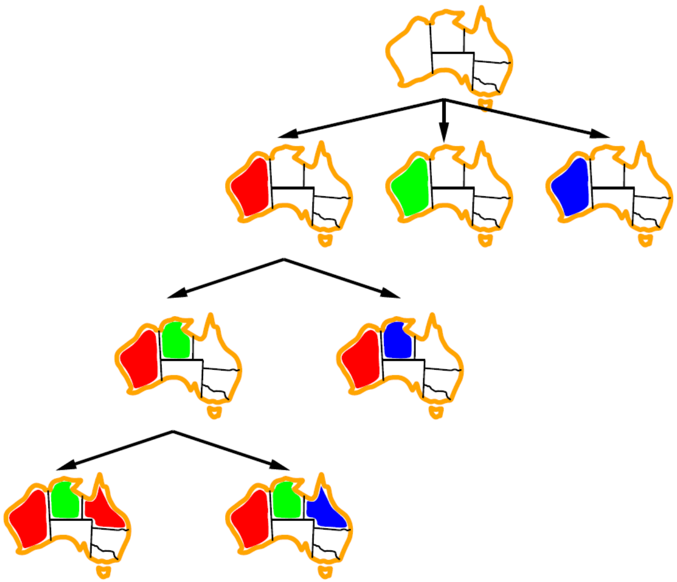
Solve a CSP as a Search Problem

- State Space
  - Each state: an **assignment** of values to some or all of the variables $\{X_i{=}v_i, X_j{=}v_j, \cdots\}$
    - Consistent assignment: does not violate any constraints
    - Complete assignment: every variable is assigned a value
    - Partial assignment: only some of the variables have been assigned values



Solve a CSP as a Search Problem

- Actions
  - An action is to assign a value to an un-assigned variable



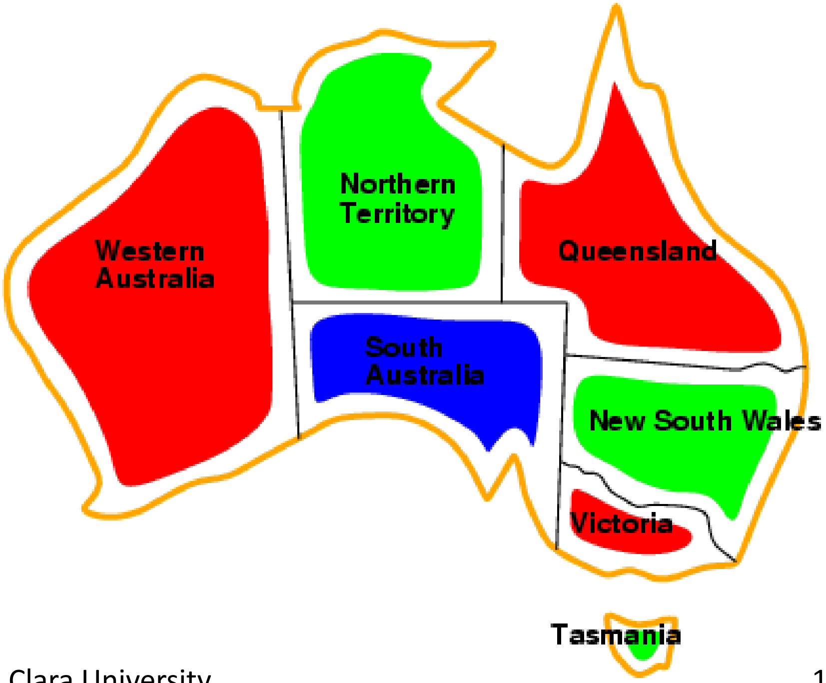
Solve a CSP as a Search Problem

- A Goal State of a CSP
  - A complete & consistent assignment

  Such as

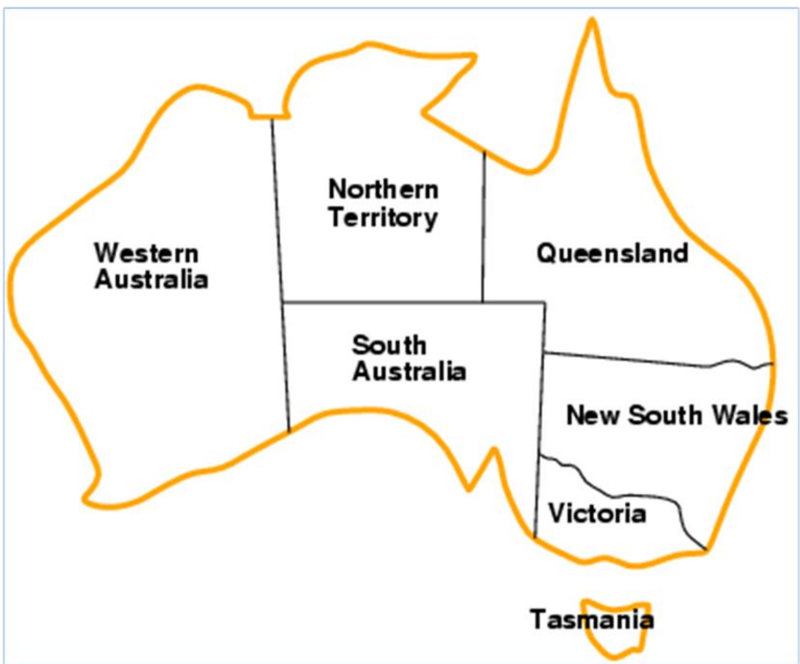
  {WA=red, NT=green, Q=red, NSW=green, V=red, SA=blue, T=green}

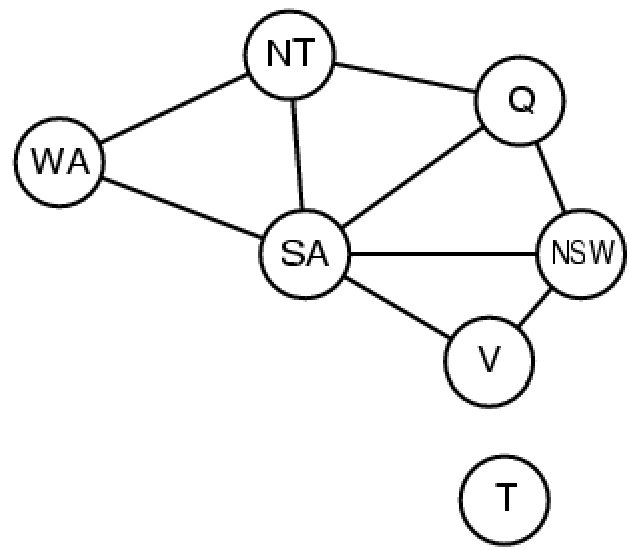
- Goal Test
  - To check whether the current assignment is consistent and complete



Constraint Graph

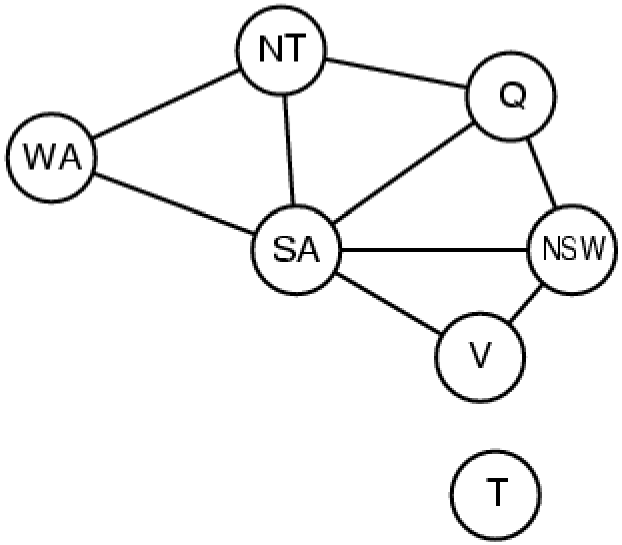
- **Nodes**: variables
- **Arcs (Links)**: constraints
  - An arc connects two variables that participate in a constraint





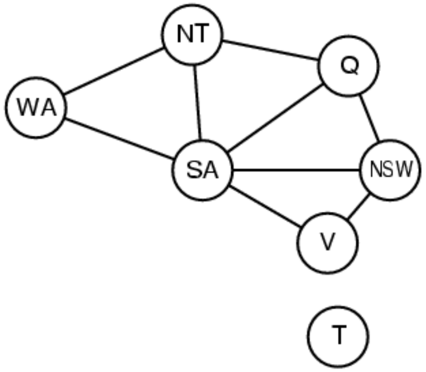
Variations of CSPs

- **Unary** constraints
  - Restrict the value of a single variable
  - Example: SA$\neq$ Green
- **Binary** constraints
  - Constraints between two variables
  - Example: SA $\neq$ WA
  - A **Binary CSP**:
    - Only has binary constraints



Variations of CSPs

- **Higher-order** constraints
  - involve 3 or more variables
  - Example: SA ≠ WA ≠ NT, i.e. *Alldiff*(SA,WA,NT)
- Global constraints (a general case)
  - Involve an arbitrary number of variables
  - Need not involve all variables
  - Example: *Alldiff*
    - All of the variables involved in the constraint must have different values



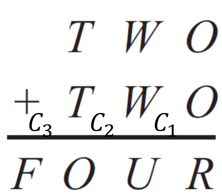
Cryptarithmetic Puzzles

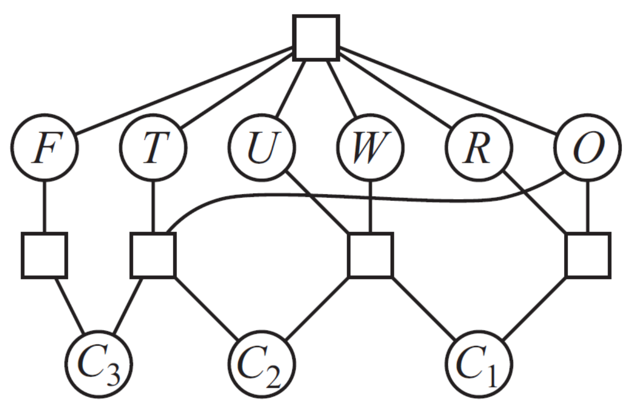
- Each letter represents a distinct digit (0-9)
  - Global constraint: square box at the top

$$Alldiff(F, T, U, W, R, O)$$

  - Four column constraints

$$O + O = R + 10 \cdot C_1$$
$$C_1 + W + W = U + 10 \cdot C_2$$
$$C_2 + T + T = O + 10 \cdot C_3$$
$$C_3 = F$$





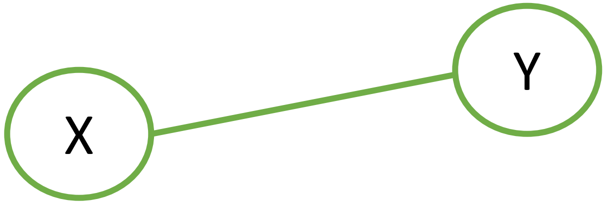
Constraint Propagation

- **Arc consistency:** a variable in a CSP is **arc-consistent** if
  - Every value in its domain satisfies the variable's binary constraints with other variables

  - Formal definition:
  - Variable X_i is arc-consistent with respect to (w.r.t.) another variable X_j if for every value in the current domain D_i, there is some value in the domain D_j that satisfies the binary constraint on the arc(X_i, X_j).

Constraint Propagation

- A network is **arc-consistent** if
  - Every variable is arc-consistent with every other variable
- Example



Domain of X and Y: digits (0 to 9)

Constraint: $Y = X^2$

- How to check the arc-consistency of this network?

Constraint Propagation

- D(X) = {0,1,2,3,4,5,6,7,8,9}
- D(Y) = {0,1,2,3,4,5,6,7,8,9}
- Constraint: $Y = X^2$

- Need to check two arcs: (X,Y) and (Y,X)

- Check (X,Y)
  - Update: D(X) = {0,1,2,3}
- Check (Y,X)
  - Update: D(Y) = {0,1,4,9}

AC-3 Algorithm

- The most popular algorithm that makes every variable arc-consistent in a CSP
- Maintains a queue (indeed, a set) of arcs to consider
- Step 1: Initially
  - The queue contains all arcs in the CSP
    - Each binary constraint becomes 2 arcs, one in each direction
    - e.g. arc (X_i, X_j) means arc $X_i \to X_j$

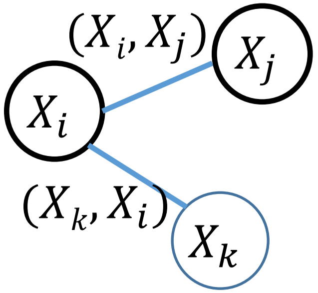
      arc (X_j, X_i) means arc $X_j \to X_i$

AC-3 Algorithm

- Step 2: Check the arcs in the queue one by one, until the queue is empty
  - Pops off an arc(X_i, X_j) from the queue
  - Makes X_i arc-consistent with respect to X_j
    - If D_i is changed when we check the arc consistency of (X_i, X_j): add to the queue all arcs (X_k, X_i) where X_k is a neighbor of X_i; $k \neq j$

      i.e. add to the queue the incoming arcs to X_i except (X_j, X_i)
    - If D_i is revised down to nothing:

      Returns failure



- AC-3 reduces the domains of variables
  - State space is smaller for the search algorithm

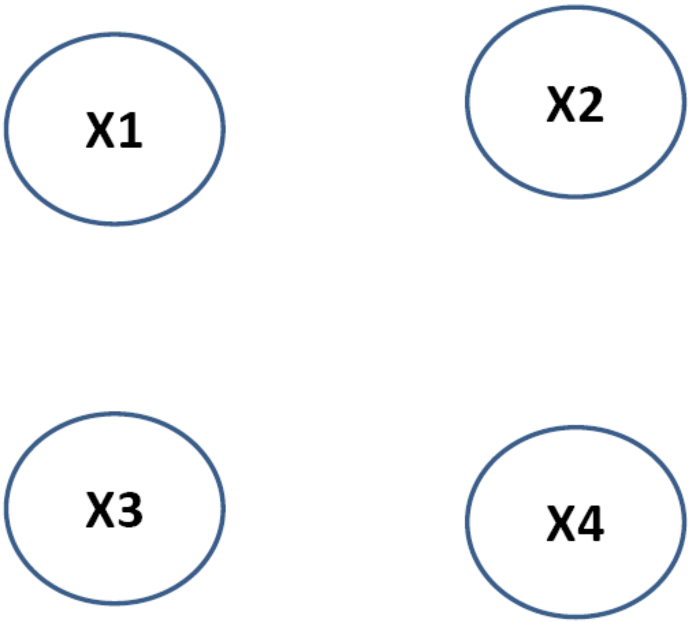
AC-3 Example

- Consider the following binary constraint network:
  - 4 variables: X1, X2, X3, X4
  - Domains:
    - D1 = {1,2,3,4}
    - D2= {3,4,5,8,9}
    - D3 = {2,3,5,6,7,9}
    - D4= {3,5,7,8,9}
  - Constraints:
    - $X1 \geq X2$
    - $X2 > X3$ or $X3 - X2 = 2$
    - $X3 \neq X4$

AC-3 Example

- a). Draw the constraint graph

4 variables: X1, X2, X3, X4 => **4 nodes**



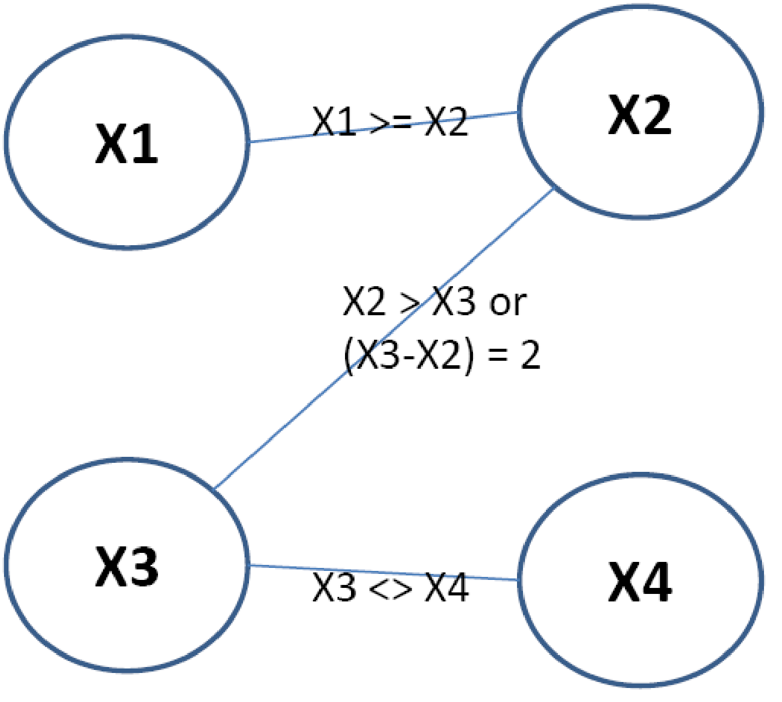
AC-3 Example

- a). Draw the constraint graph

4 variables: X1, X2, X3, X4 => **<u>4 nodes</u>**

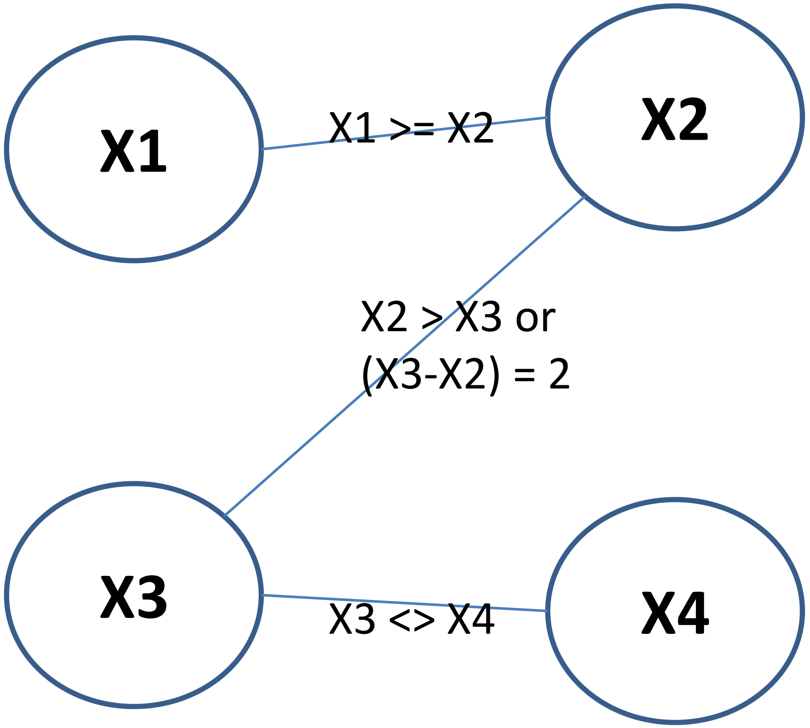
Constraints:

- X1 >=X2
- X2 > X3 or
- (X3 – X2) = 2
- X3 <> X4



AC-3 Example

- b). Is the network arc consistent? If not, compute the arc-consistent network using the AC-3 algorithm
- Arc X→Y is consistent iff for every value x of X there is some allowed y of Y
  - X1=1<D2={3,4,5,8,9}
  - Not arc consistent!



Domains:

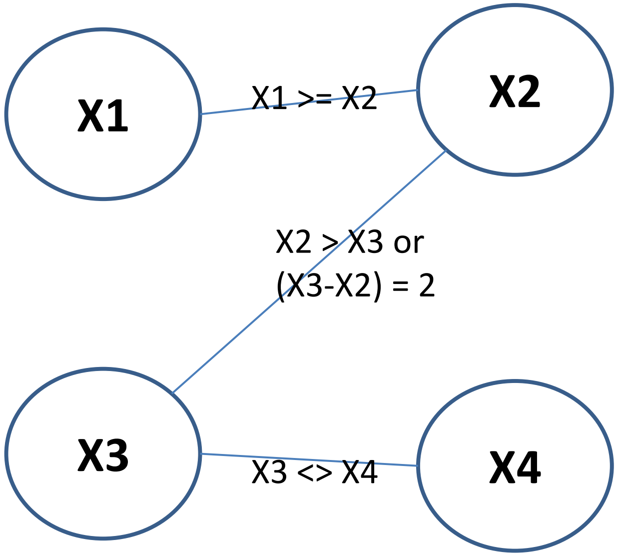
D1 = {1,2,3,4}
D2= {3,4,5,8,9}
D3 = {2,3,5,6,7,9}
D4= {3,5,7,8,9}

AC-3 Example

- Run AC-3 Algorithm
- Queue of arcs:

(X1, X2), (X3,X2), (X2,X3), (X4, X3), (X2,X1), (X3,X4)

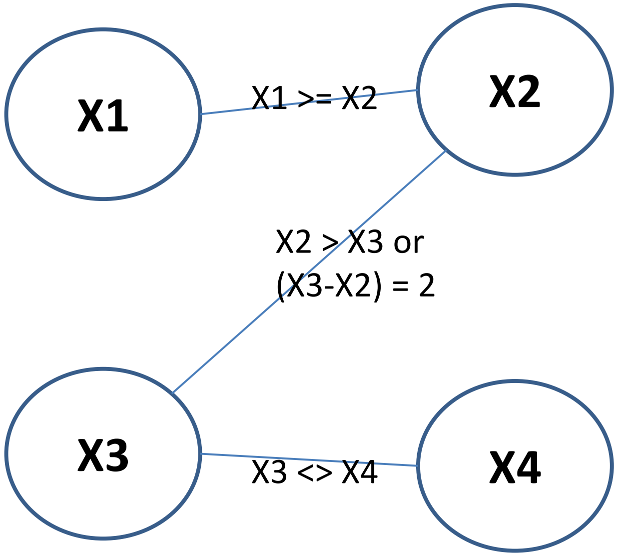
- Current Domain
- D1={1,2,3,4}
- D2={3,4,5,8,9}
- D3={2,3,5,6,7,9}
- D4={3,5,7,8,9}



AC-3 Example

- Run AC-3 Algorithm
- Queue of arcs:

  (X1,X2), (X3,X2), (X2,X3), (X4, X3), (X2,X1), (X3,X4)
- Check (X1,X2)
  - Constraint: X1>=X2
- Current Domain
- D1={1,2,3,4}
- D2={3,4,5,8,9}
- D3={2,3,5,6,7,9}
- D4={3,5,7,8,9}



- Result: Remove {1,2} from D1

AC-3 Example

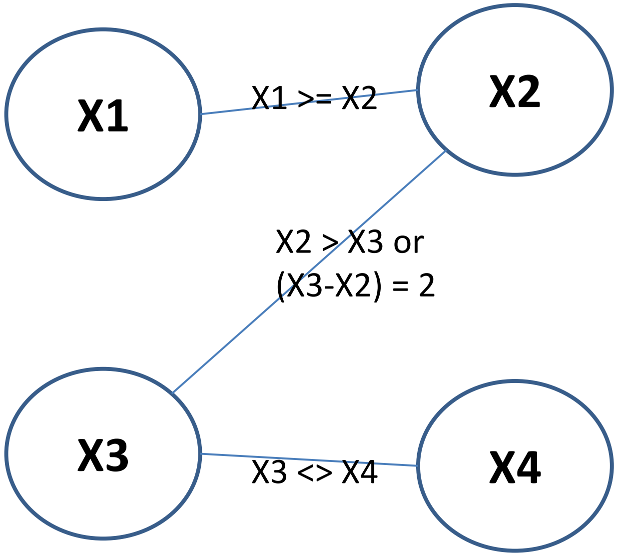
Recall: AC-3

If D_i is changed while checking arc consistency of (X_i, X_j): add to the queue all arcs (X_k, X_i) where X_k is a neighbor of X_i, $k \neq j$ (i.e. incoming arcs to X_i except that from X_j)

- Run AC-3 Algorithm
- Queue of arcs:

(X1,X2), (X3,X2), (X2,X3), (X4, X3), (X2,X1), (X3,X4)

- Check (X1,X2)
  - Constraint: X1>=X2
- Current Domain
- D1={3,4}
- D2={3,4,5,8,9}
- D3={2,3,5,6,7,9}
- D4={3,5,7,8,9}

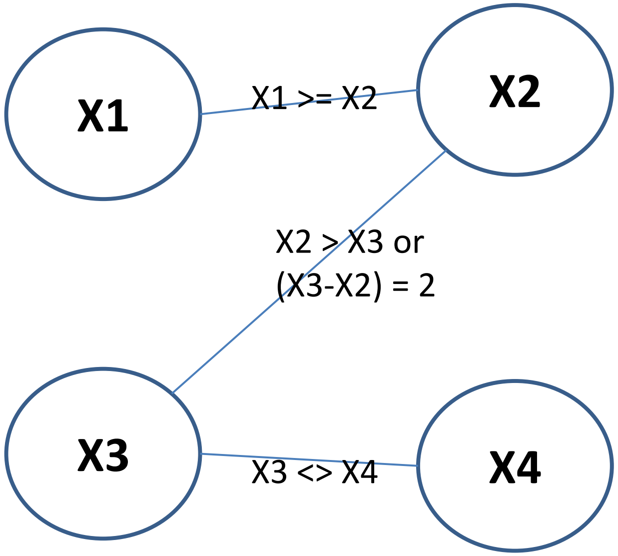


- Result: Remove {1,2} from D1

AC-3 Example

- Run AC-3 Algorithm
- Queue of arcs:

  (X3,X2), (X2,X3), (X4, X3), (X2,X1), (X3,X4)
- Check (X3,X2)
  - Constraints: X2>X3 or (X3-X2)=2
- Current Domain
- D1={3,4}
- D2={3,4,5,8,9}
- D3={2,3,5,6,7,9}
- D4={3,5,7,8,9}
- Result: Remove {9} from D3



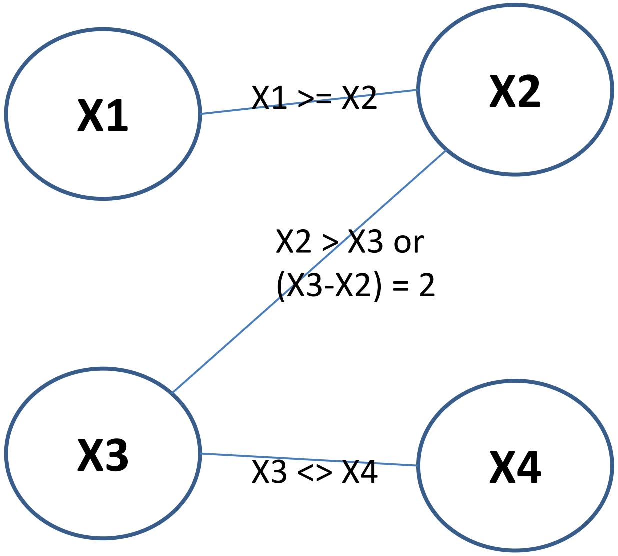
AC-3 Example

Recall: AC-3

If D_i is changed while checking arc consistency of (X_i, X_j) : add to the queue all arcs (X_k, X_i) where X_k is a neighbor of X_i, $k \neq j$ (i.e. incoming arcs to X_i except that from X_j)

- Run AC-3 Algorithm
- Queue of arcs:

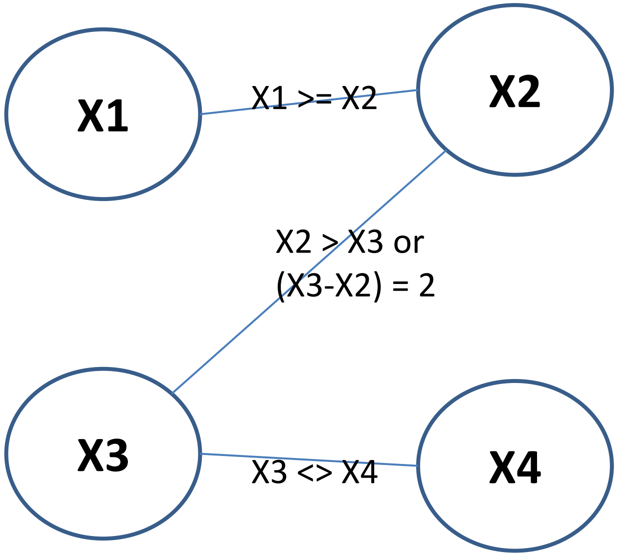
  (X3,X2), (X2,X3), (X4, X3), (X2,X1), (X3,X4)
- Check (X3,X2)
  - Constraints: X2>X3 or (X3-X2)=2
- Current Domain
- D1={3,4}
- D2={3,4,5,8,9}
- D3={2,3,5,6,7}
- D4={3,5,7,8,9}
- Result: Remove {9} from D3



AC-3 Example

- Run AC-3 Algorithm
- Queue of arcs:

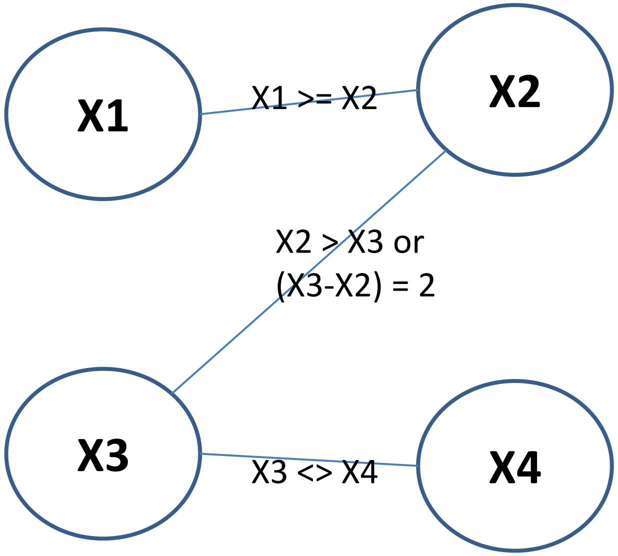
  (X2,X3), (X4, X3), (X2,X1), (X3,X4)
- Check (X2,X3)
  - Constraints: X2>X3 or (X3-X2)=2
- Current Domain
- D1={3,4}
- D2={3,4,5,8,9}
- D3={2,3,5,6,7}
- D4={3,5,7,8,9}
- Result: do nothing



AC-3 Example

- Run AC-3 Algorithm
- Queue of arcs:

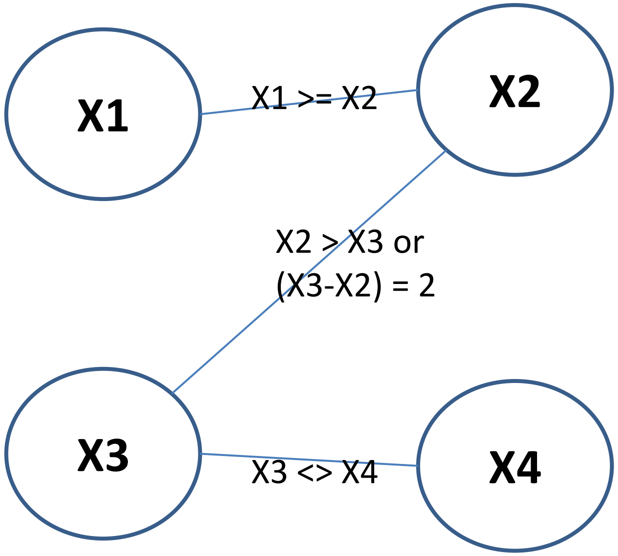
  (X4,X3), (X2,X1), (X3,X4)
- Check (X4,X3)
  - Constraints: X3<>X4
- Current Domain
- D1={3,4}
- D2={3,4,5,8,9}
- D3={2,3,5,6,7}
- D4={3,5,7,8,9}
- Result: do nothing



AC-3 Example

- Run AC-3 Algorithm
- Queue of arcs:

  (X2,X1), (X3,X4)
- Check (X2,X1)
  - Constraints: X1>=X2
- Current Domain
- D1={3,4}
- D2={3,4,5,8,9}
- D3={2,3,5,6,7}
- D4={3,5,7,8,9}
- Result: Remove {5,8,9} from D2

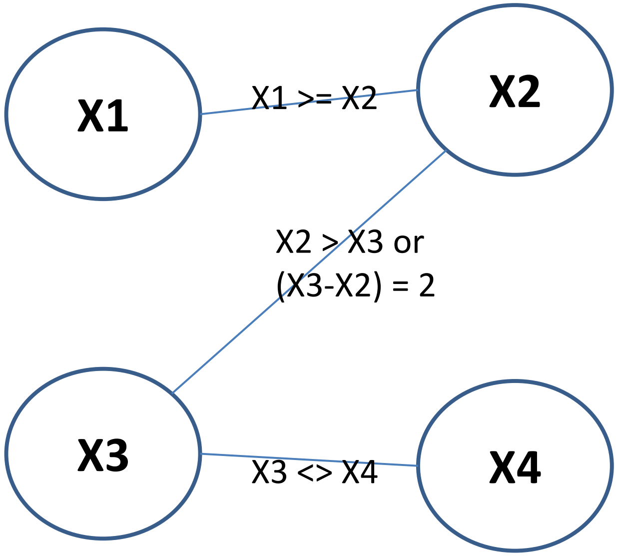


AC-3 Example

- Run AC-3 Algorithm
- Queue of arcs:
  (X2,X1), (X3,X4)
- Check (X2,X1)
  - Constraints: X1>=X2
- Current Domain
- D1={3,4}
- D2={3,4}
- D3={2,3,5,6,7}
- D4={3,5,7,8,9}
- Result: Remove {5,8,9} from D2

Recall: AC-3

If D_i is changed while checking arc consistency of (X_i, X_j) : add to the queue all arcs (X_k, X_i) where X_k is a neighbor of X_i, $k \neq j$ (i.e. incoming arcs to X_i except that from X_j)

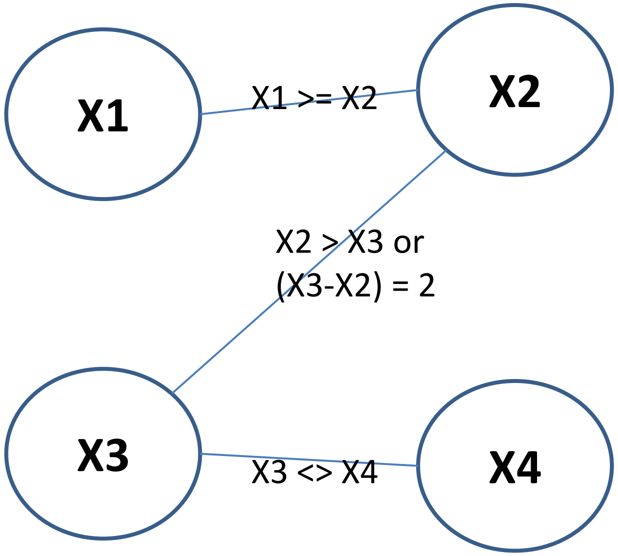


AC-3 Example

- Run AC-3 Algorithm
- Queue of arcs:
  (X2,X1), (X3,X4), (X3,X2)
- Check (X2,X1)
  - Constraints: X1>=X2
- Current Domain
- D1={3,4}
- D2={3,4}
- D3={2,3,5,6,7}
- D4={3,5,7,8,9}
- Result: Remove {5,8,9} from D2,
- add (X3,X2) to queue

Recall: AC-3

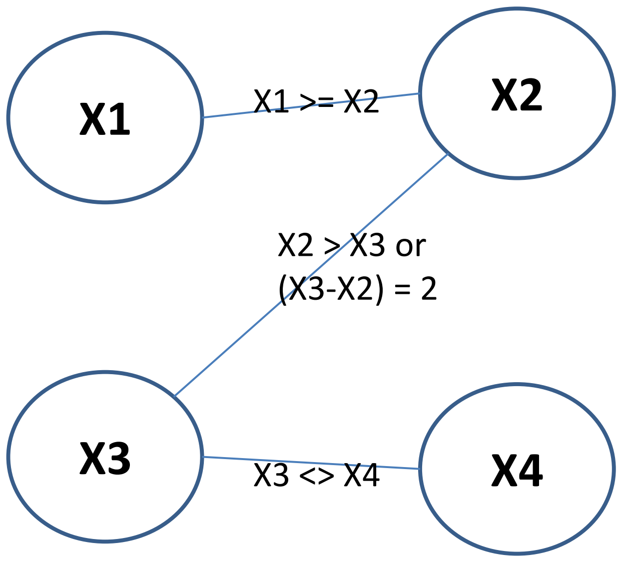
If D_i is changed while checking arc consistency of (X_i, X_j) : add to the queue all arcs (X_k, X_i) where X_k is a neighbor of X_i, $k \neq j$ (i.e. incoming arcs to X_i except that from X_j)



AC-3 Example

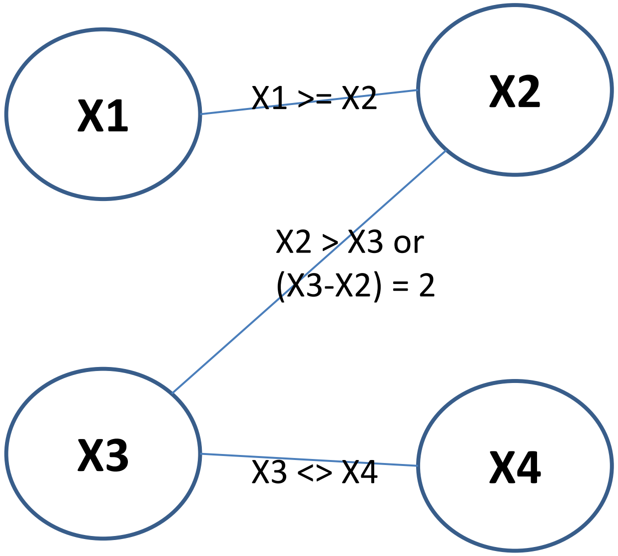
- Run AC-3 Algorithm
- Queue of arcs:

  (X3,X4), (X3,X2)
- Check (X3,X4)
  - Constraints: X3<>X4
- Current Domain
- D1={3,4}
- D2={3,4}
- D3={2,3,5,6,7}
- D4={3,5,7,8,9}
- Result: do nothing
-



AC-3 Example

- Run AC-3 Algorithm
- Queue of arcs:
  (X3,X2)
- Check (X3,X2)
  - Constraints: X2>X3 or (X3-X2)=2
- Current Domain
- D1={3,4}
- D2={3,4}
- D3={2,3,5,6,7}
- D4={3,5,7,8,9}
- Result: remove {7} from D3
-



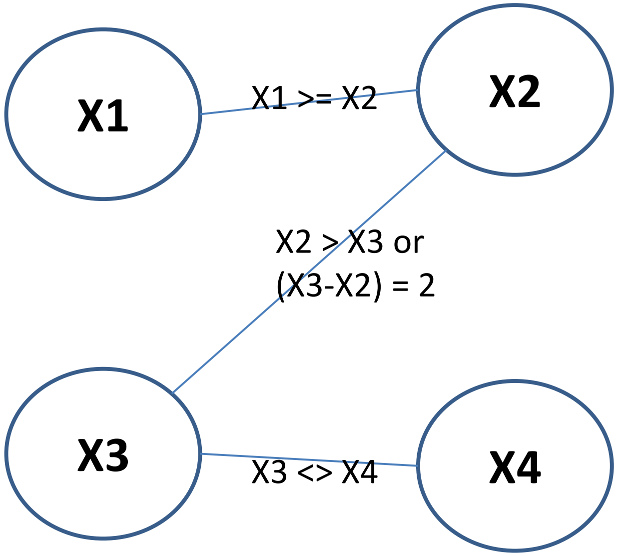
AC-3 Example

- Run AC-3 Algorithm
- Queue of arcs:

  (X3,X2)
- Check (X3,X2)
  - Constraints: X2>X3 or (X3-X2)=2
- Current Domain
- D1={3,4}
- D2={3,4}
- D3={2,3,5,6}
- D4={3,5,7,8,9}
- Result: remove {7} from D3
-

Recall: AC-3

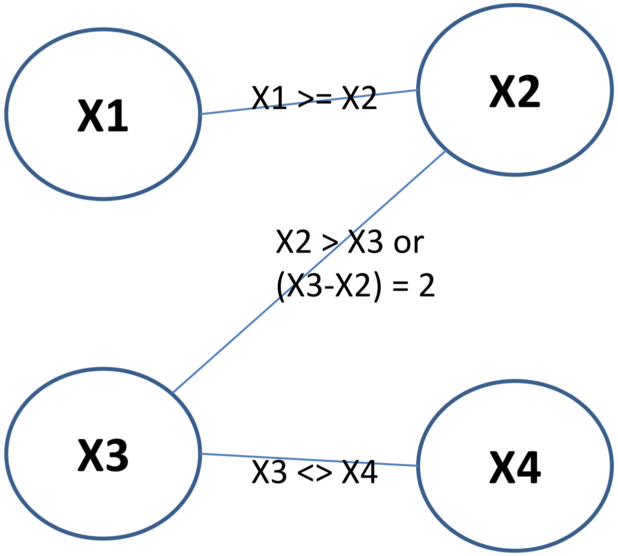
If D_i is changed while checking arc consistency of (X_i, X_j) : add to the queue all arcs (X_k, X_i) where X_k is a neighbor of X_i, $k \neq j$ (i.e. incoming arcs to X_i except that from X_j)



AC-3 Example

- Run AC-3 Algorithm
- Queue of arcs:

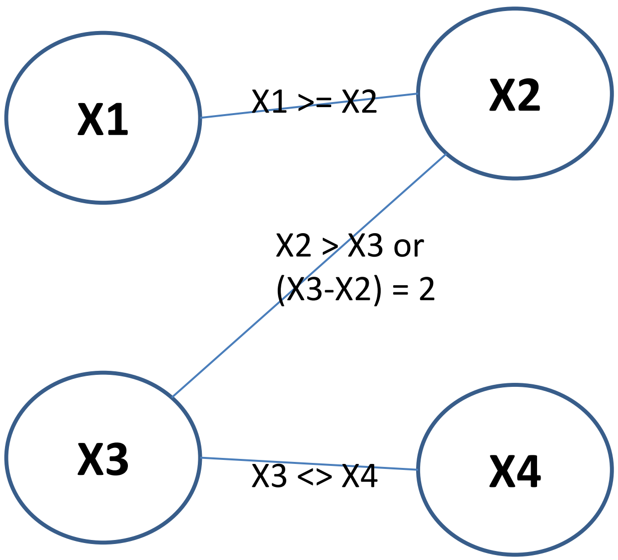
  (X3,X2), (X4,X3)
- Check (X3,X2)
  - Constraints: X2>X3 or (X3-X2)=2
- Current Domain
- D1={3,4}
- D2={3,4}
- D3={2,3,5,6}
- D4={3,5,7,8,9}
- Result: remove {7} from D3,
- add (X4,X3) to queue



AC-3 Example

- Run AC-3 Algorithm
- Queue of arcs:

  (X4,X3)
- Check (X4,X3)
  - Constraints: X3<>X4
- Current Domain
- D1={3,4}
- D2={3,4}
- D3={2,3,5,6}
- D4={3,5,7,8,9}
- Result: do nothing

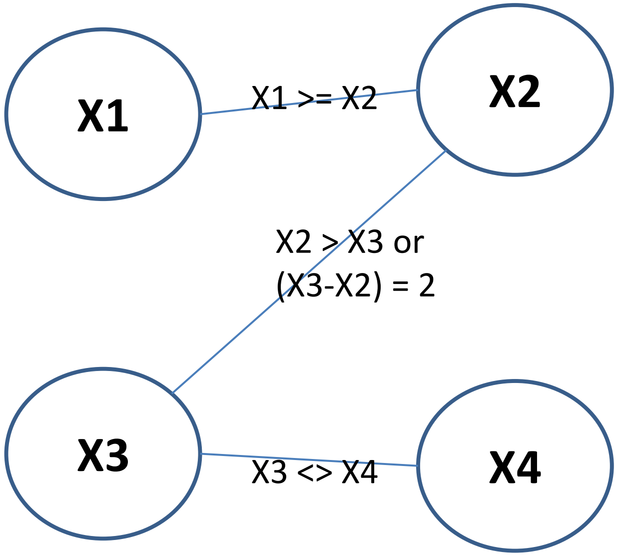


AC-3 Example

- Run AC-3 Algorithm
- Queue of arcs:

  None

- Reduced Domain (Arc Consistent)
- D1={3,4}
- D2={3,4}
- D3={2,3,5,6}
- D4={3,5,7,8,9}



AC-3 Example

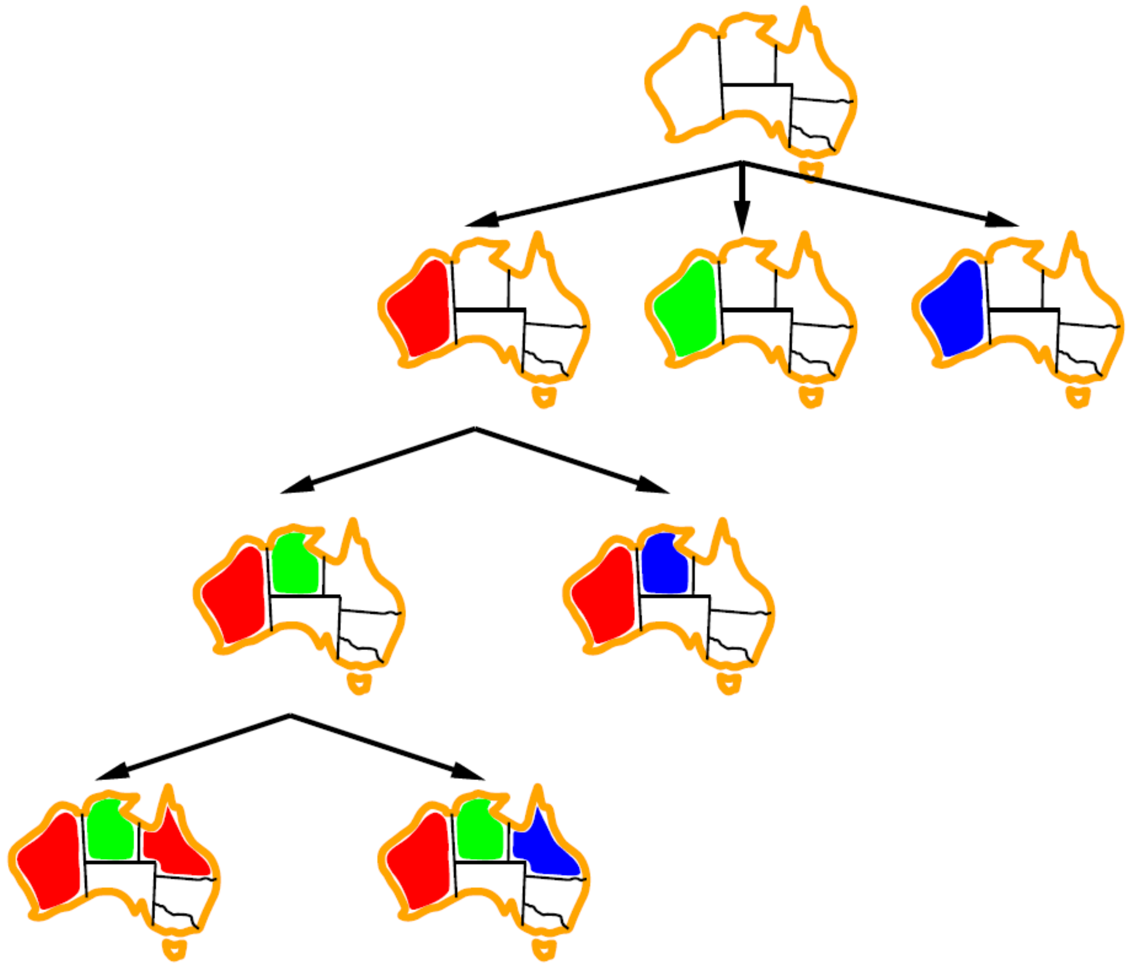
- C. Is the network arc consistent? If yes, give a solution.

- 4 variables: X1, X2, X3, X4
- Constraints:
  - X1 >=X2
  - X2 > X3 or (X3 – X2) = 2
  - X3 <> X4
- Network is arc-consistent for domains:
  - D1’ = {3, 4}
  - D2’ = {3, 4}
  - D3’ = {2, 3, 5, 6}
  - D4’ = {3, 5, 7, 8, 9}

- One of possible solutions is:

X1 = 3; X2 = 3; X3 = 2; X4 = 3

Backtracking Search

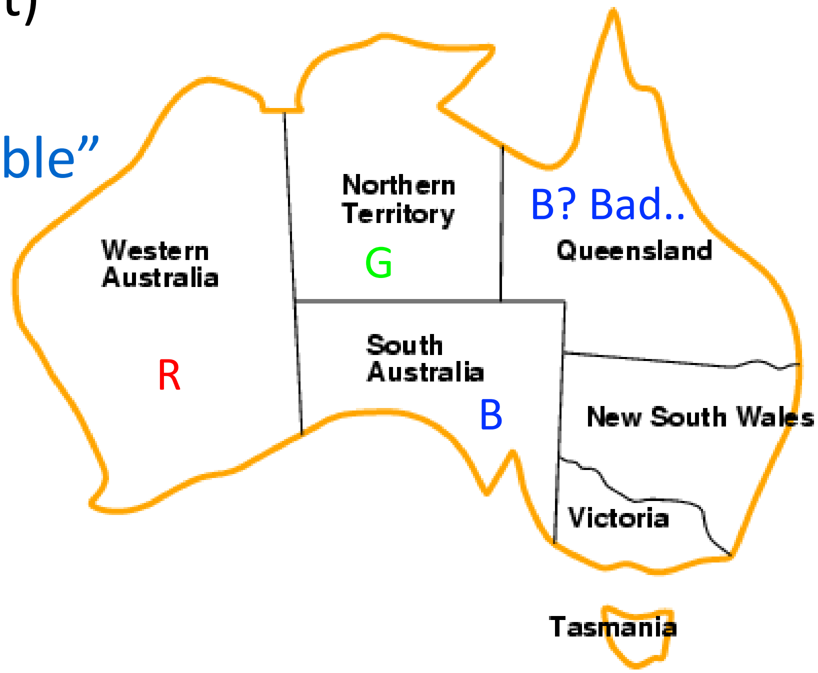


Variable Ordering

- Which variable should be assigned next?
  - SELECT-UNASSIGNED-VARIABLE
- Static Ordering
  - Chooses the next unassigned variable in order, $\{X_1, X_2, \ldots\}$
  - Seldom results in the most efficient search

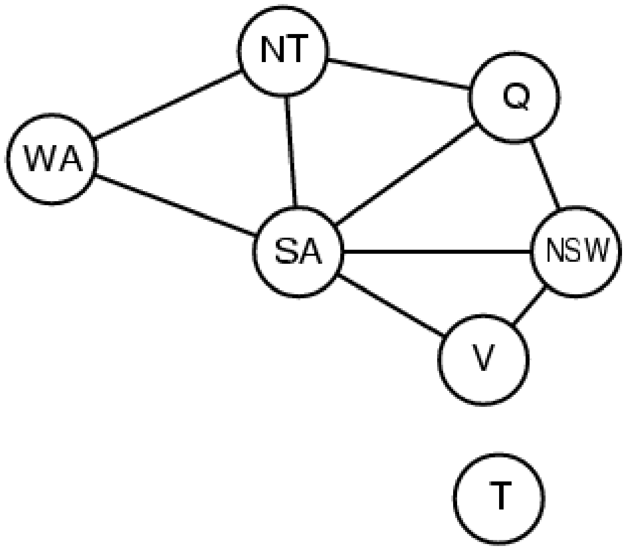
MRV heuristic

- **Minimum-Remaining-Values (MRV) heuristic:** Chooses the variable with the fewest "legal" values
  - Example:
    - WA=red, NT=green, then
    - Only one possible value for SA, SA=blue
    - (rather than assigning Q next)
  - Also called
    - The "most constrained variable"
  - Usually performs better than a random or static ordering



Degree Heuristic

- The MRV heuristic doesn't help at all in choosing the first region to color in Australia
- **Degree heuristic:** selects the variable that is involved in the largest number of constraints
  - Example: SA has the highest degree, 5; other variables have degree 2 or 3 (exception: T has 0)



Value Ordering

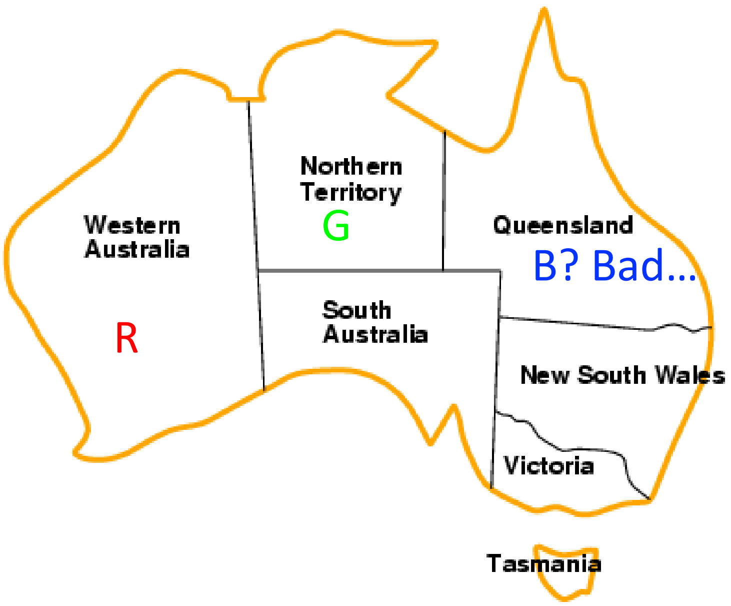
- Once a variable has been selected, in what order should its values be tried?
- The **least-constraining-value** heuristic:
  – prefers the value that rules out the fewest choices for the neighboring variables in the constraint graph

Value Ordering

- **least-constraining-value** heuristic:
- Example: WA=red, NT=green, and our next choice is for Q
  - Q=blue: bad choice, eliminates the last legal value left for Q's neighbor SA.
  - The least-constraining-value heuristic:

    Prefers Q=red

    Tries to leave the maximum flexibility for subsequent variable assignments



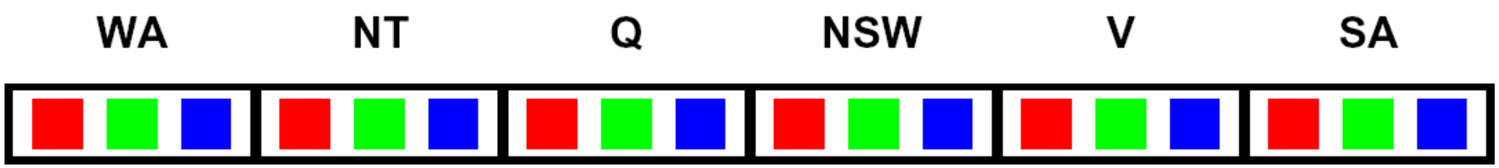
- Rule of thumb: **fail-last**

Forward Checking

- When solving a CSP, we can apply inference methods
- Purpose: infer reductions of the domain for other variables
- Forward Checking
  - Keep track of remaining legal values for unassigned variables that are connected to current variable.
  - Terminate search (Backtrack) when any variable has no legal values

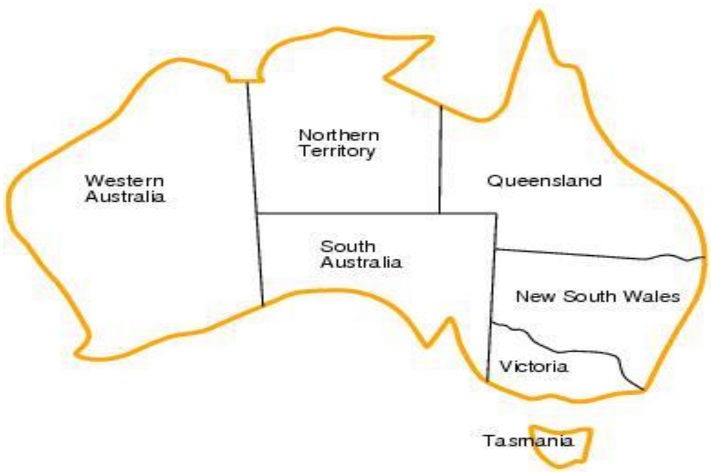
Forward Checking

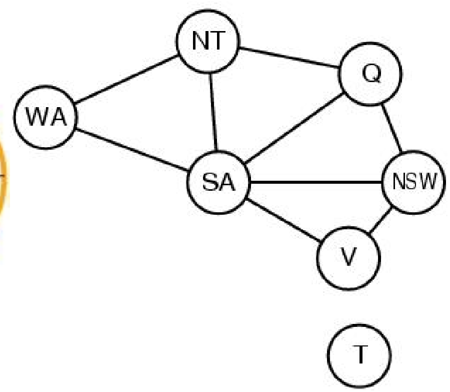
	WA	NT	Q	NSW	V	SA	T
Initial domains	R G B	R G B	R G B	R G B	R G B	R G B	R G B
After *WA=red*	Ⓡ	G B	R G B	R G B	R G B	G B	R G B
After *Q=green*	Ⓡ	B	Ⓖ	R B	R G B	B	R G B
After *V=blue*	Ⓡ	B	Ⓖ	R	Ⓑ		R G B



Example

- a. Forward checking: NT has been assigned a value as shown. Cross out all values that would be eliminated by Forward Checking.





WA = Western Australia
NT = Northern Territory
Q = Queensland
SA = South Australia
NSW = New South Wales
V = Victoria
T = Tasmania

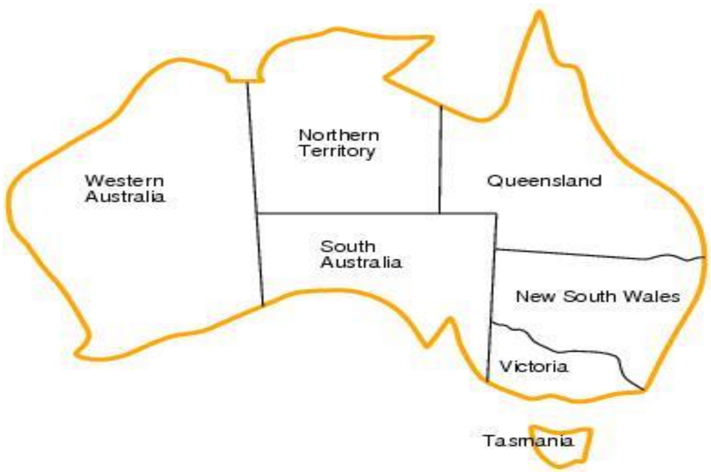
WA	NT	Q	SA	NSW	V	T
R G B	G	R G B	R G B	R G B	R G B	R G B

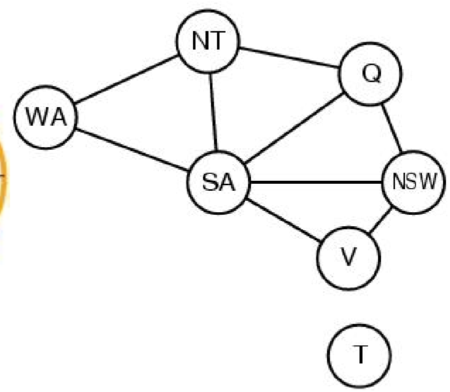
- Answer:

WA	NT	Q	SA	NSW	V	T
R ~~G~~ B	G	R ~~G~~ B	R ~~G~~ B	R G B	R G B	R G B

Example

- b. Minimum-remaining-value (MRV) heuristic: consider the assignment below. WA is assigned and forward checking has been done. List all unassigned variables that might be selected by the MRV heuristic





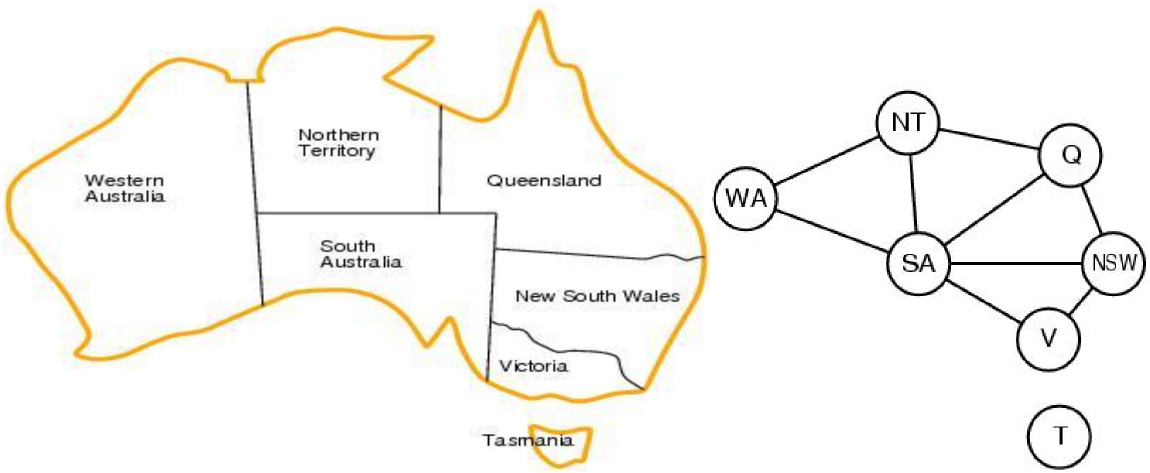
WA = Western Australia
NT = Northern Territory
Q = Queensland
SA = South Australia
NSW = New South Wales
V = Victoria
T = Tasmania

WA	NT	Q	SA	NSW	V	T
R	G B	R G B	G B	R G B	R G B	R G B

- Answer:
- NT, SA

Example

- c. Degree heuristic: consider the assignment below. WA is assigned and forward checking has been done. List all unassigned variables that might be selected by the Degree heuristic



WA = Western Australia
NT = Northern Territory
Q = Queensland
SA = South Australia
NSW = New South Wales
V = Victoria
T = Tasmania

WA	NT	Q	SA	NSW	V	T
R	G B	R G B	G B	R G B	R G B	R G B

- Answer:
- SA

Local Search for CSP

- Use a complete-state formulation
  - the initial state has assigned a value to every variable
  - The search changes the value of one variable at a time
- Example: the 8-queens problem
  - the initial state might be a random configuration of 8 queens in 8 columns
    - the initial assignment violates several constraints
  - each step moves a single queen to a new position in its column
- The point of local search: to eliminate the violated constraints

Local Search for CSP

- The min-conflicts heuristic
  - In choosing a new value for a variable, select the value that results in the minimum number of conflicts with other variables

Min-Conflicts Heuristic

- Example

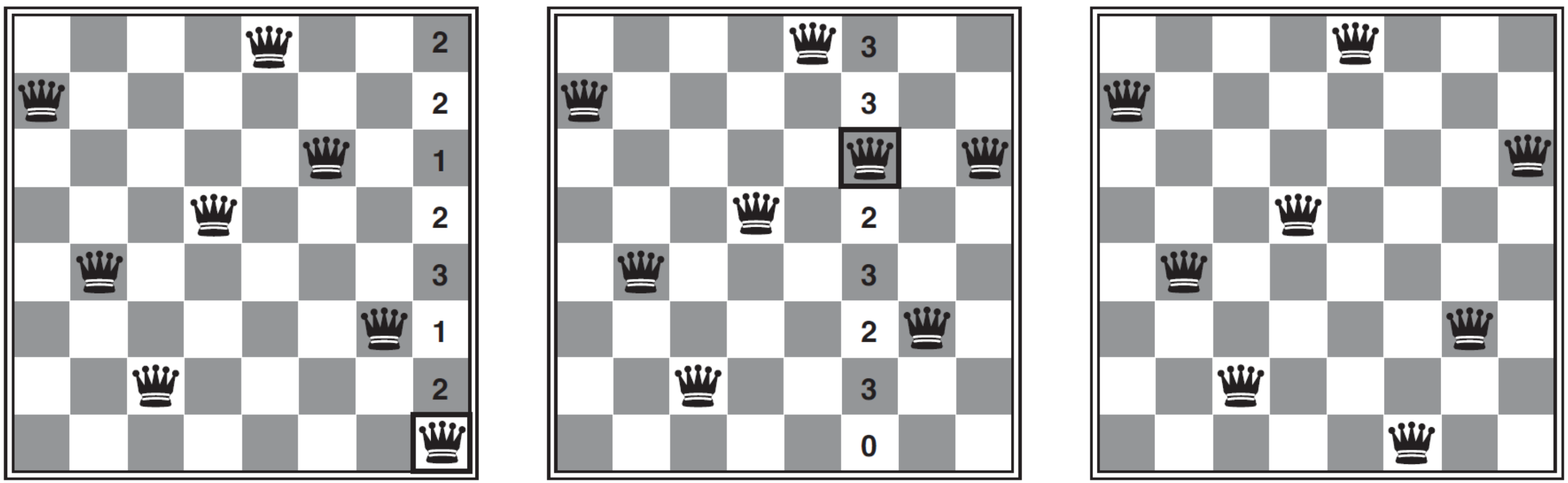
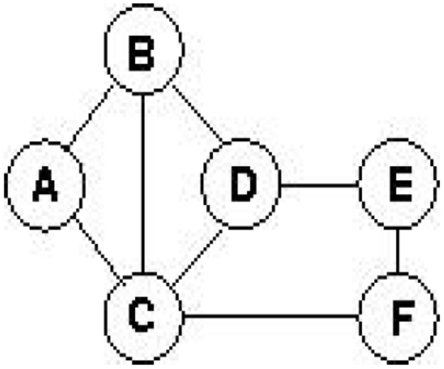


Figure 6.9 A two-step solution using min-conflicts for an 8-queens problem. At each stage, a queen is chosen for reassignment in its column. The number of conflicts (in this case, the number of attacking queens) is shown in each square. The algorithm moves the queen to the min-conflicts square, breaking ties randomly.

Example

You are given the following constraint graph representing a map that has to be colored with three colors, red (R), green (G) and blue (B), subject to the constraints that no adjacent regions, which are connected by an arc in the graph, are assigned the same color.



Consider the set of inconsistent assignments below. Variable **C** has just been selected to be assigned a new value during a local search for a complete and consistent assignment. What new *value* will be selected for variable **C** by the **Min-Conflicts Heuristic**?

A	B	C	D	E	F
B	G		G	G	B

Answer: R