Constraint Satisfaction Problems

A Quick Overview
(based on AIMA book slides)

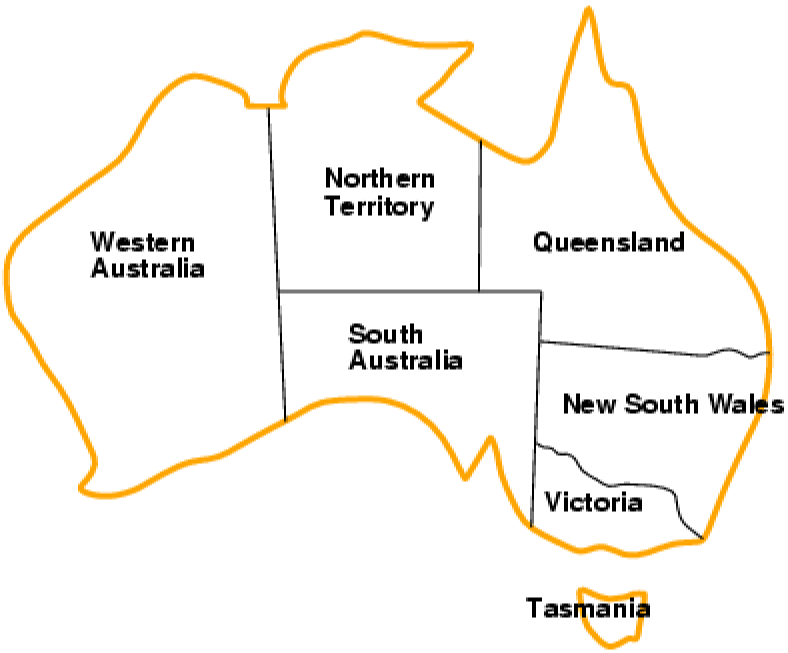
Constraint satisfaction problems

- **What is a CSP?**
  - *Finite set of variables* $V_1, V_2, \ldots, V_n$
  - *Nonempty domain of possible values* for each variable $D_{V1}, D_{V2}, \ldots D_{Vn}$
  - *Finite set of constraints* $C_1, C_2, \ldots, C_m$
  - Each *constraint* C_i limits the values that variables can take, e.g., $V_1 \neq V_2$
- **A *state* is an *assignment* of values to some or all variables.**
- ***Consistent assignment:* assignment does not violate the constraints.**

Constraint satisfaction problems

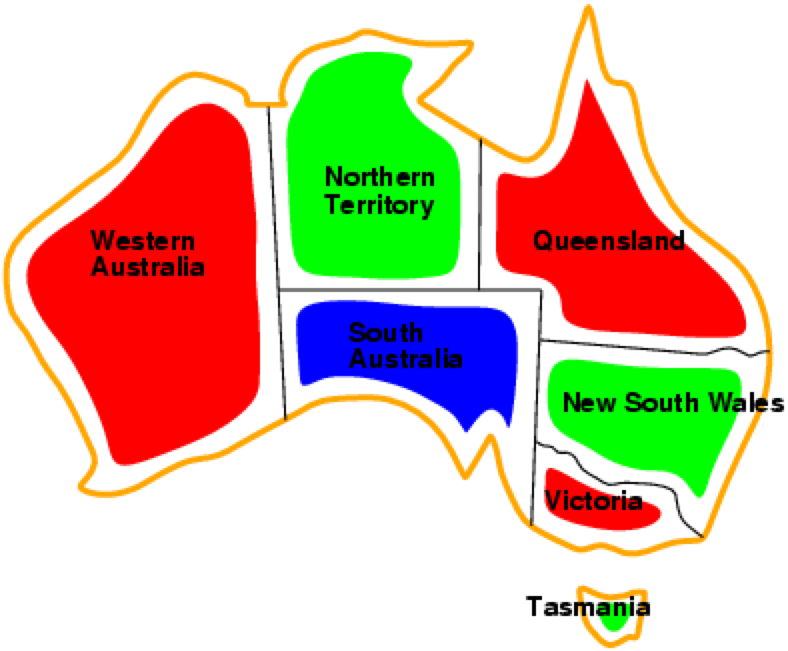
- **An assignment is *complete* when every variable has a value.**
- **A *solution* to a CSP is a complete assignment that satisfies all constraints.**
- **Some CSPs require a solution that maximizes an *objective function*.**
- **Applications:**
  - Scheduling the Hubble Space Telescope,
  - Floor planning for VLSI,
  - Map coloring,
  - Cryptography

Example: Map-Coloring



- **Variables:** ***WA, NT, Q, NSW, V, SA, T***
- **Domains:** D_i **= {red,green,blue}**
- **Constraints: adjacent regions must have different colors**
  - e.g., WA ≠ NT
    - —So (WA,NT) must be in {(red,green),(red,blue),(green,red), …}

Example: Map-Coloring

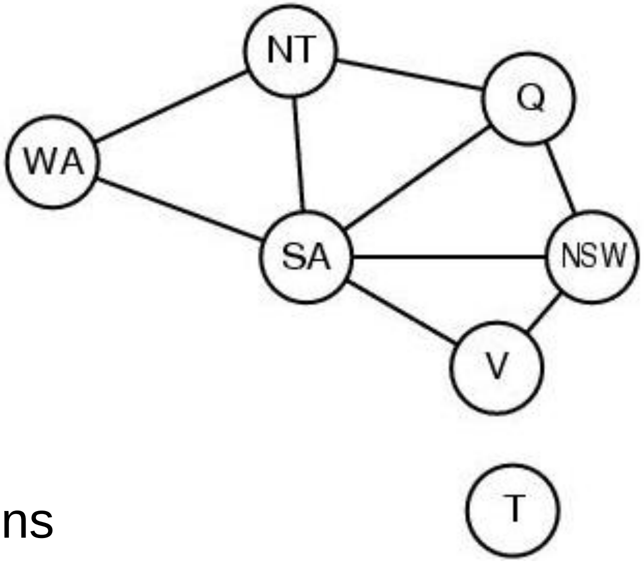


Solutions are complete and consistent assignments,

- e.g., WA = red, NT = green,Q = red,NSW = green,
  V = red,SA = blue,T = green

Constraint graph

- ***Binary CSP:*** **each constraint relates two variables**
- ***Constraint graph:***
  - *nodes* are variables
  - *arcs* are constraints
- **CSP benefits**
  - Standard representation pattern
  - Generic goal and successor functions
  - Generic heuristics (no domain specific expertise).
- **Graph can be used to simplify search.**
  - —**e.g. Tasmania is an independent subproblem**.



Varieties of CSPs

- **Discrete variables**
  - finite domains:
    - —n variables, domain size $d \rightarrow O(d^n)$ complete assignments
    - —e.g., Boolean CSPs, includes Boolean satisfiability (NP-complete)
  - infinite domains:
    - —integers, strings, etc.
    - —e.g., job scheduling, variables are start/end days for each job
    - —need a constraint language, e.g., $StartJob_1 + 5 \leq StartJob_3$

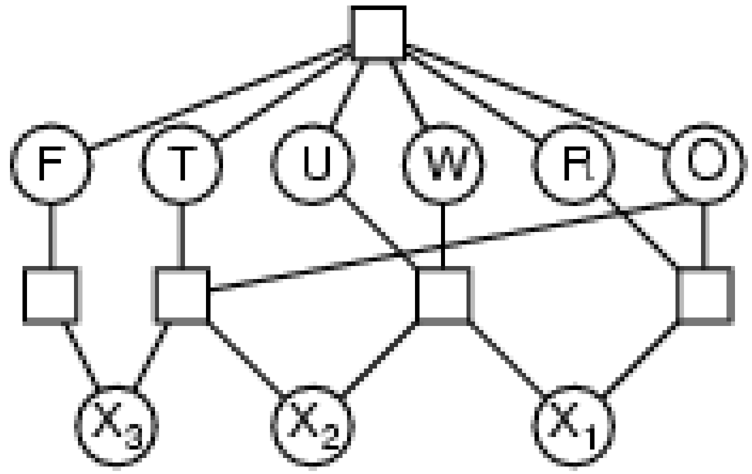
- **Continuous variables**
  - e.g., start/end times for Hubble Space Telescope observations
  - linear constraints solvable in polynomial time by linear programming

Varieties of constraints

- ***Unary*** **constraints involve a single variable,**
  - e.g., SA ≠ green
- ***Binary*** **constraints involve pairs of variables,**
  - e.g., SA ≠ WA
- ***Higher-order*** **constraints involve 3 or more variables**
  - e.g., cryptarithmetic column constraints
- ***Preference*** **(soft constraints) e.g. *red* is better than *green* can be represented by a cost for each variable assignment => Constrained optimization problems.**

Example: Cryptarithmetic

```
    T W O
  + T W O
  -------
  F O U R
```



- **Variables**: ***F T U W R O X_1 X_2 X_3***
- **Domain**: **{*0,1,2,3,4,5,6,7,8,9*}**
- **Constraints**: ***Alldiff (F,T,U,W,R,O)***
  - $O + O = R + 10 \cdot X_1$
  - $X_1 + W + W = U + 10 \cdot X_2$
  - $X_2 + T + T = O + 10 \cdot X_3$
  - $X_3 = F$, $T \neq 0$, $F \neq 0$

CSP as a standard search problem

- **A CSP can easily be expressed as a standard search problem.**
  - *Initial State:* the empty assignment {}.
  - *Operators:* Assign value to unassigned variable provided that there is no conflict.
  - *Goal test:* assignment consistent and complete.
  - *Path cost:* constant cost for every step.
  - Solution is found at depth *n*, for *n* variables
  - Hence depth first search can be used

Backtracking search

- **Variable assignments are *commutative*,**
  - Eg *[WA = red then NT = green]*
    equivalent to *[NT = green then WA = red]*

- **Only need to consider assignments to a single variable at each node**

  → $b = d$ and there are d^n leaves

- **Depth-first search for CSPs with single-variable assignments is called *backtracking* search**

- **Backtracking search basic uninformed algorithm for CSPs**

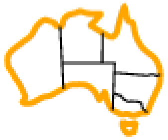
- **Can solve *n*-queens for $n \approx 25$**

Backtracking search

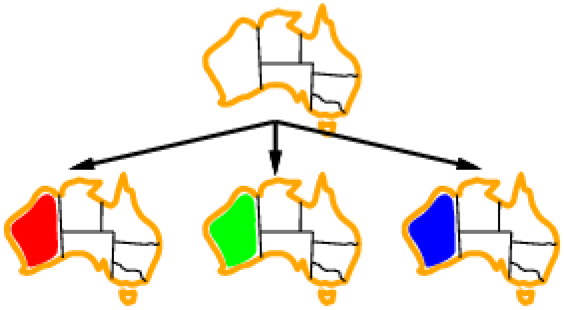
function **BACKTRACKING-SEARCH(*csp*)** % returns a solution or failure
    return **RECURSIVE-BACKTRACKING(*{} , csp*)**

function **RECURSIVE-BACKTRACKING(*assignment, csp*)** % returns a solution or failure
    if ***assignment*** **is complete** then return ***assignment***
    var $\leftarrow$ **SELECT-UNASSIGNED-VARIABLE(VARIABLES[*csp*],*assignment*,*csp*)**
    for each ***value*** in **ORDER-DOMAIN-VALUES(*var, assignment, csp*)** do
        if ***value*** **is consistent with *assignment* according to CONSTRAINTS[*csp*]** then
            add *{var=value}* to assignment
            result $\leftarrow$ **RECURSIVE-BACKTRACKING(*assignment, csp*)**
            if ***result*** $\neq$ ***failure*** then return ***result***
            remove *{var=value}* from *assignment*
    return *failure*

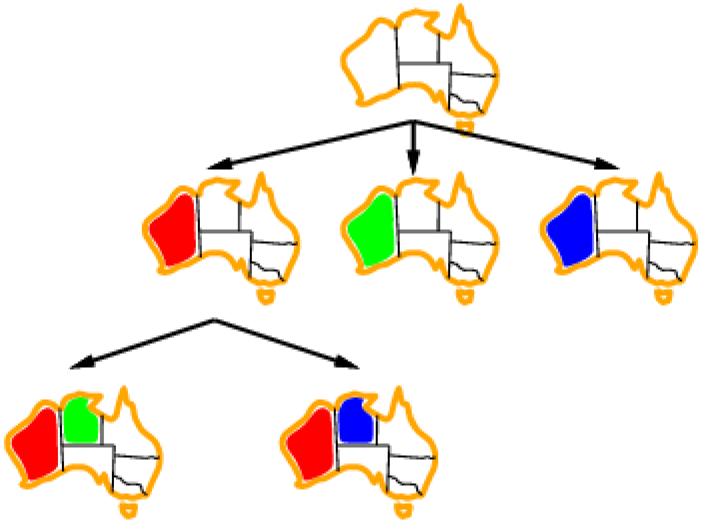
Backtracking example



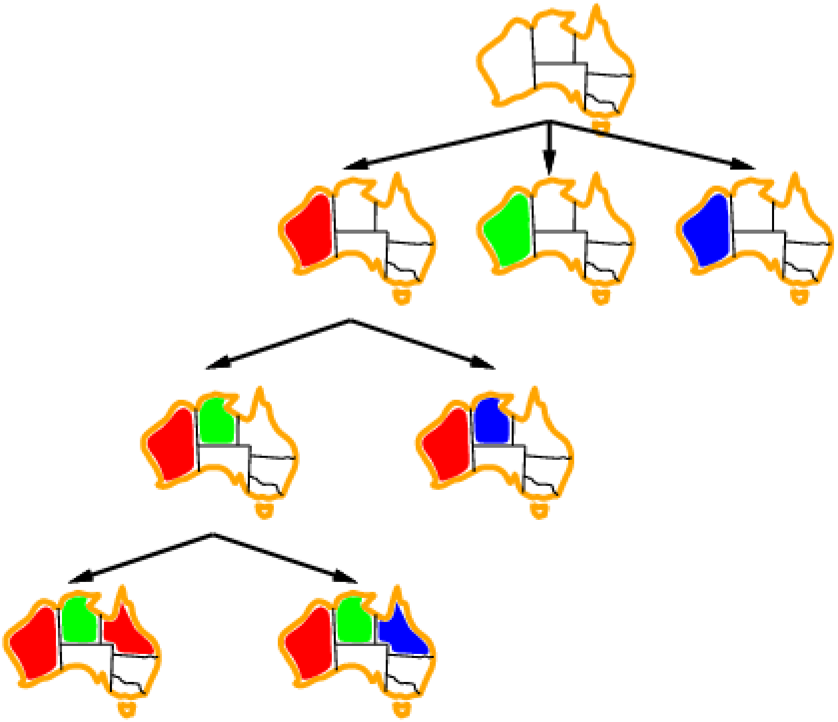
Backtracking example



Backtracking example



Backtracking example



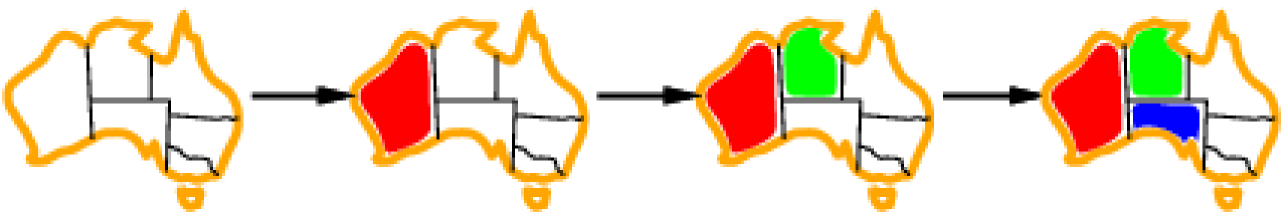
Improving backtracking efficiency

General-purpose **methods can give huge speed gains:**

- Which variable should be assigned next?
- In what order should its values be tried?
- Can we detect inevitable failure early?

Most constrained variable

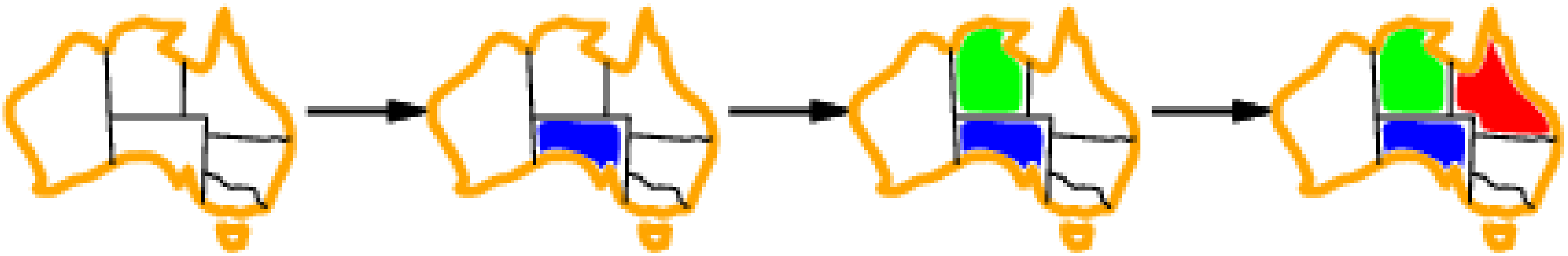
- **Most constrained variable:**
  choose the variable with the fewest legal values



- **a.k.a. *minimum remaining values (MRV)* heuristic**

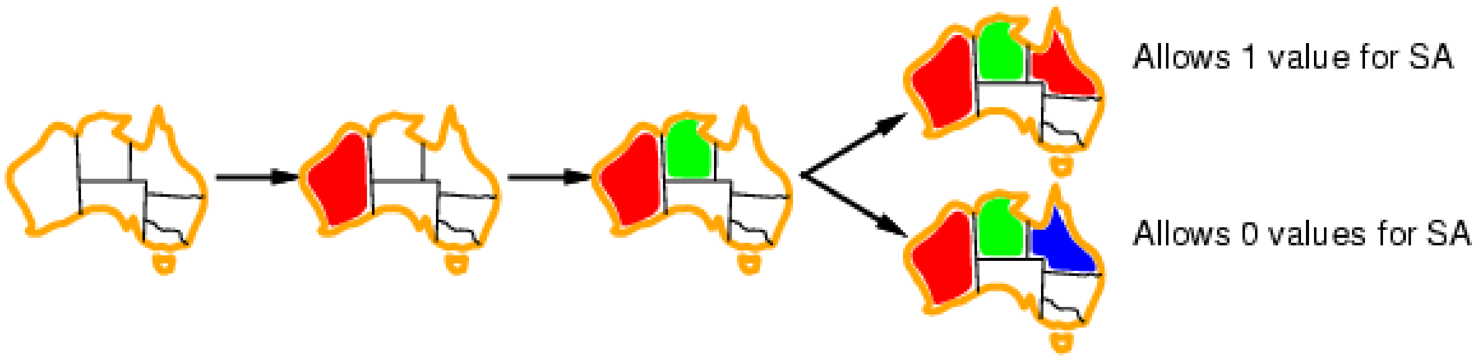
Most constraining variable

- **Tie-breaker among most constrained variables**
- **Most constraining variable:**
  - choose the variable with the most constraints on remaining variables



Least constraining value

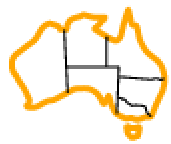
- **Given a variable, choose the least constraining value:**
  - the one that rules out the fewest values in the remaining variables

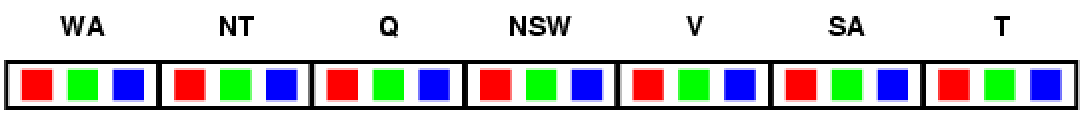


- **Combining these heuristics makes 1000 queens feasible**

Forward Checking

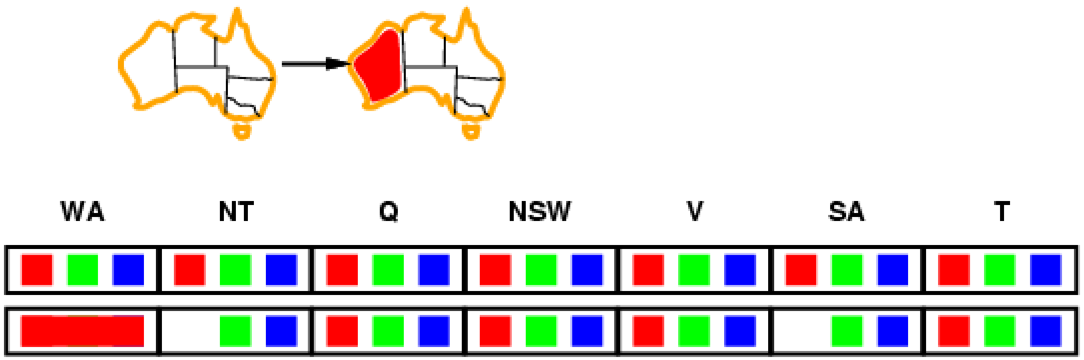
- **Idea**:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values





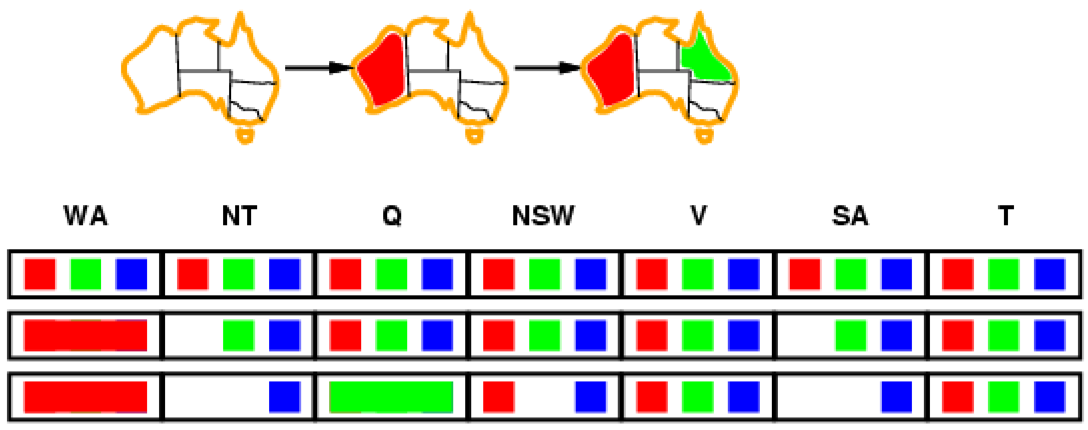
Forward checking

- **Idea**:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values



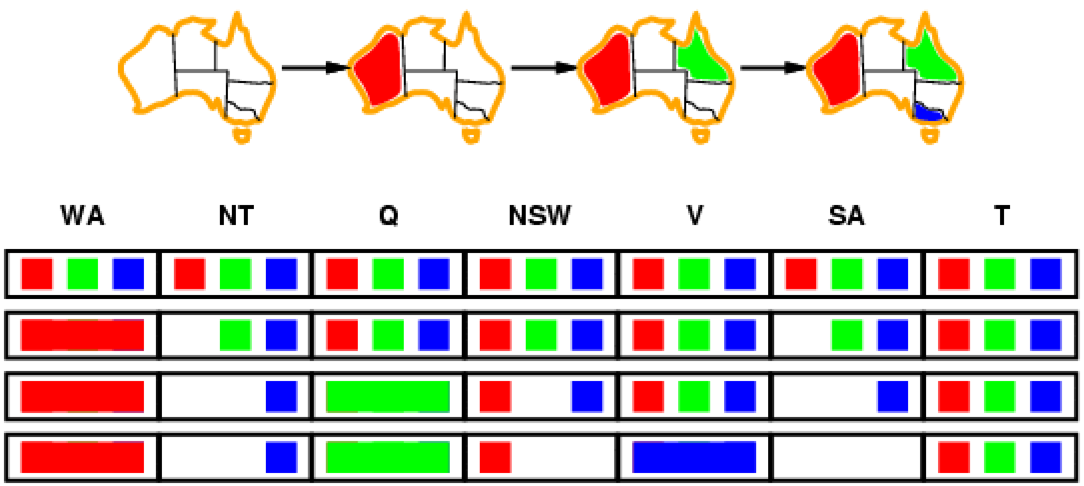
Forward checking

- **Idea**:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values



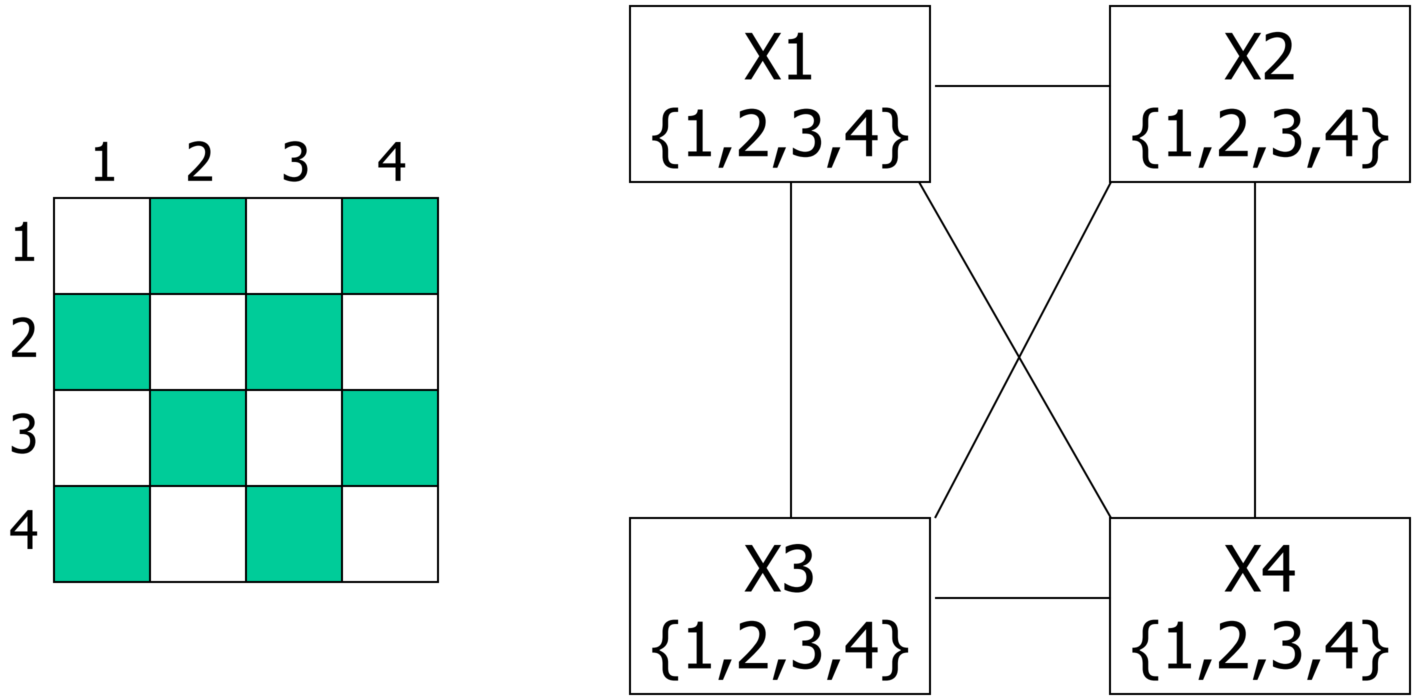
Forward checking

- **Idea:**
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values



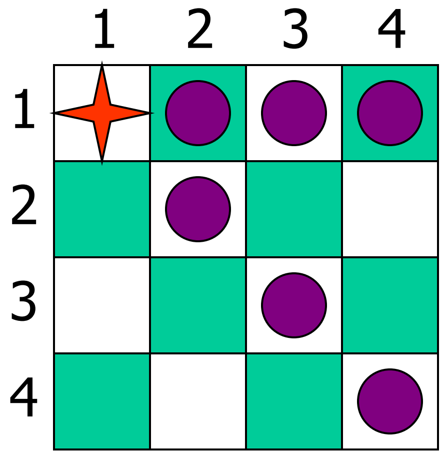
- **No more value for SA: backtrack**

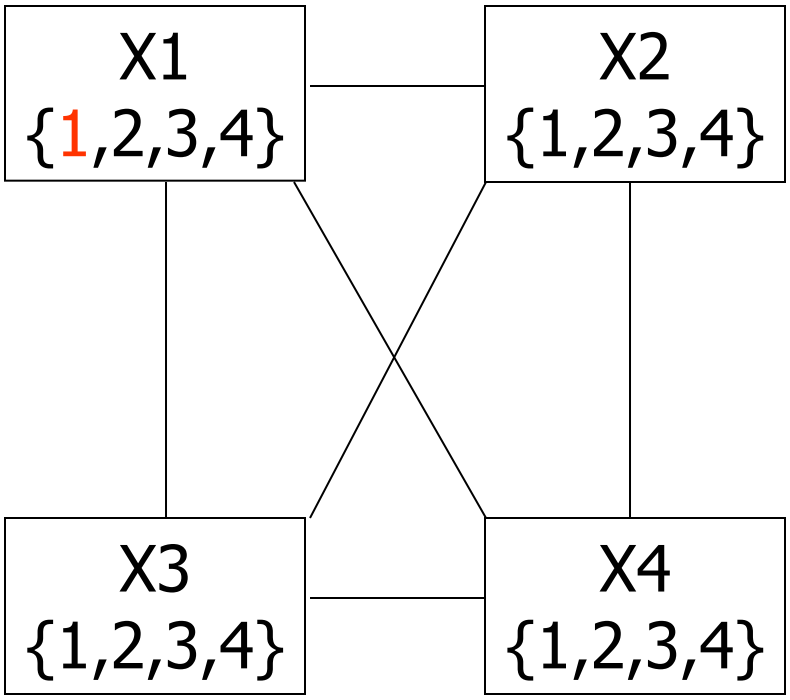
Example: 4-Queens Problem



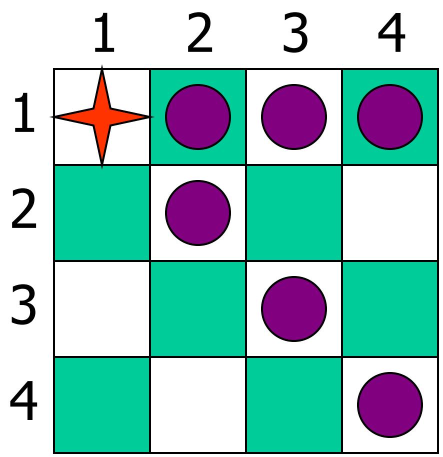
[4-Queens slides copied from B.J. Dorr]

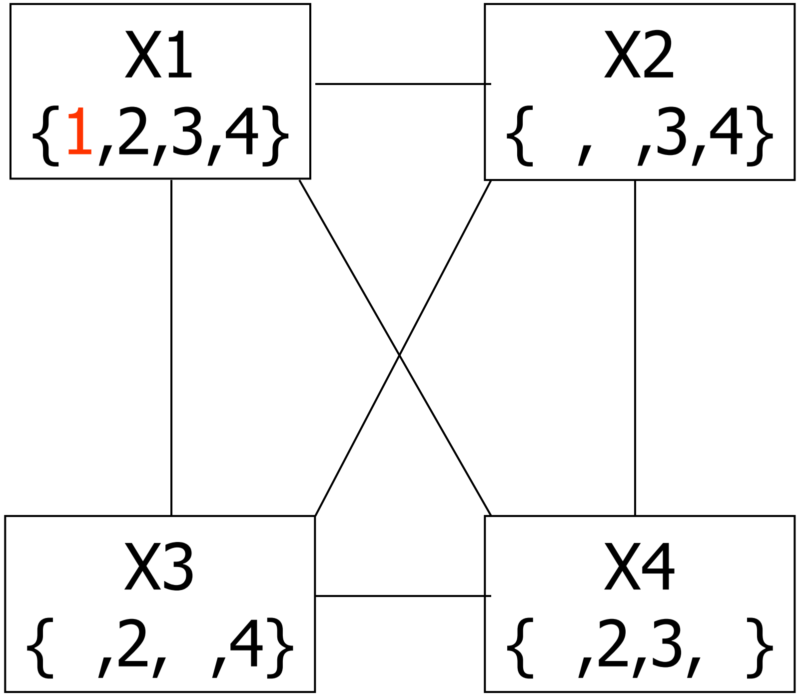
Example: 4-Queens Problem



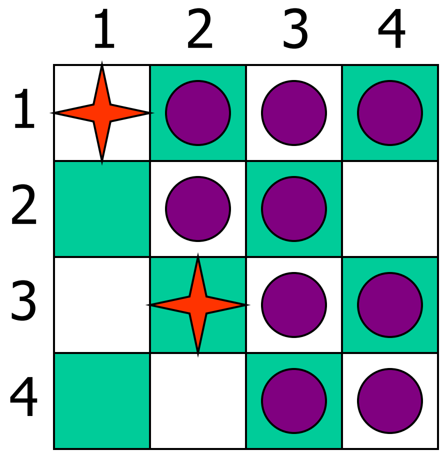


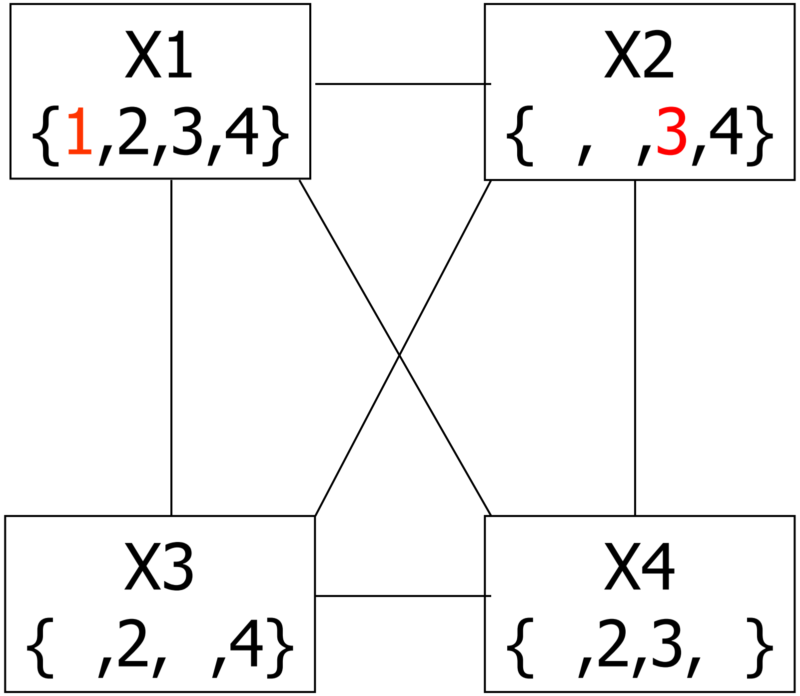
Example: 4-Queens Problem



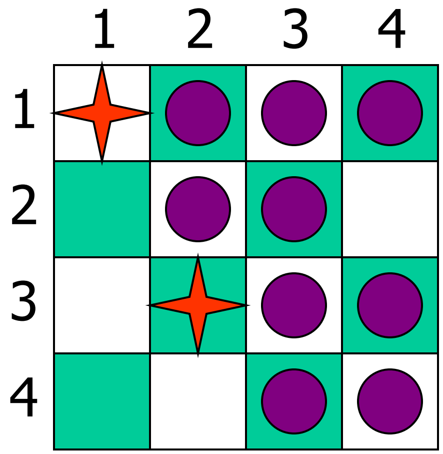


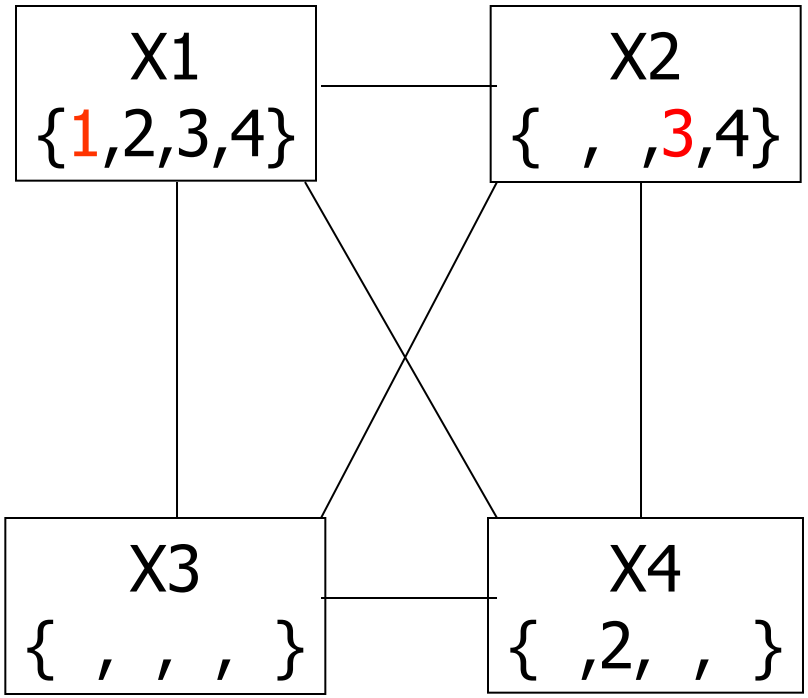
Example: 4-Queens Problem



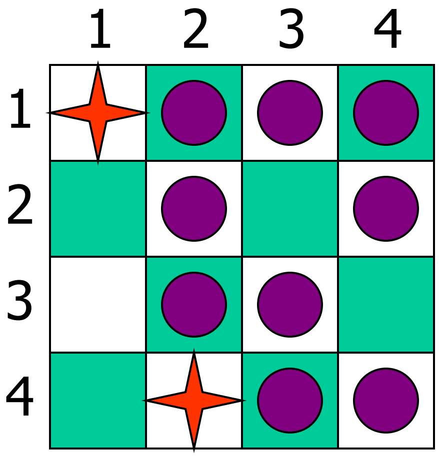


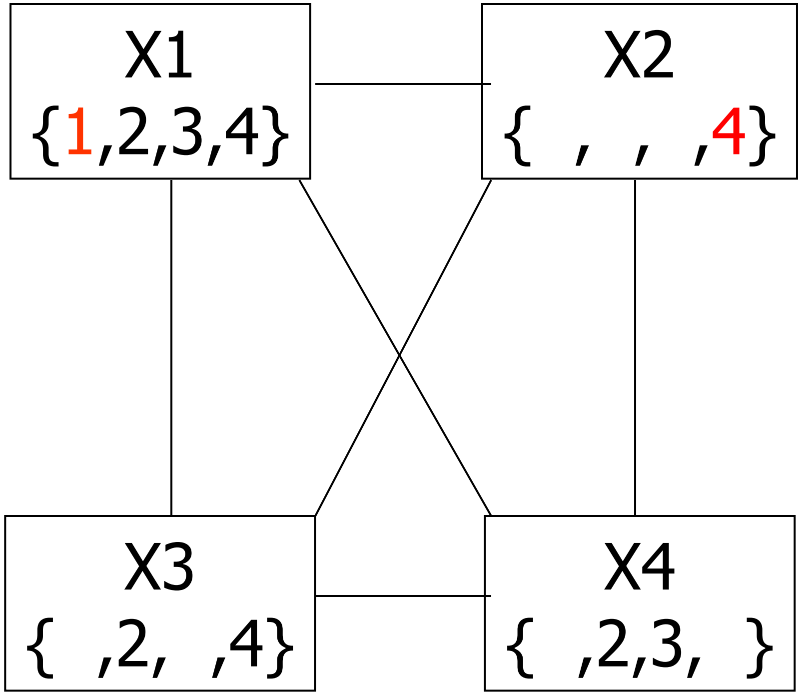
Example: 4-Queens Problem



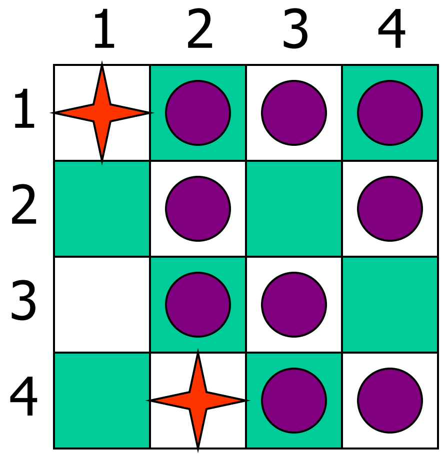


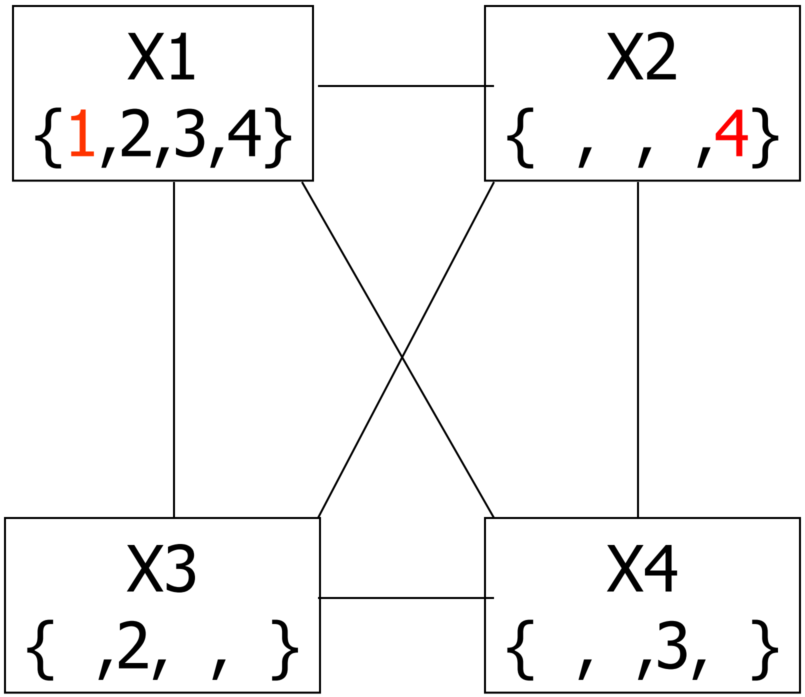
Example: 4-Queens Problem



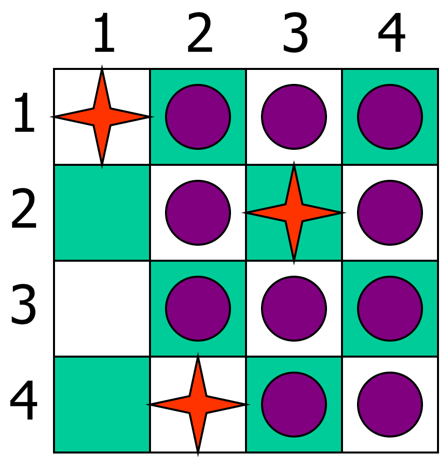


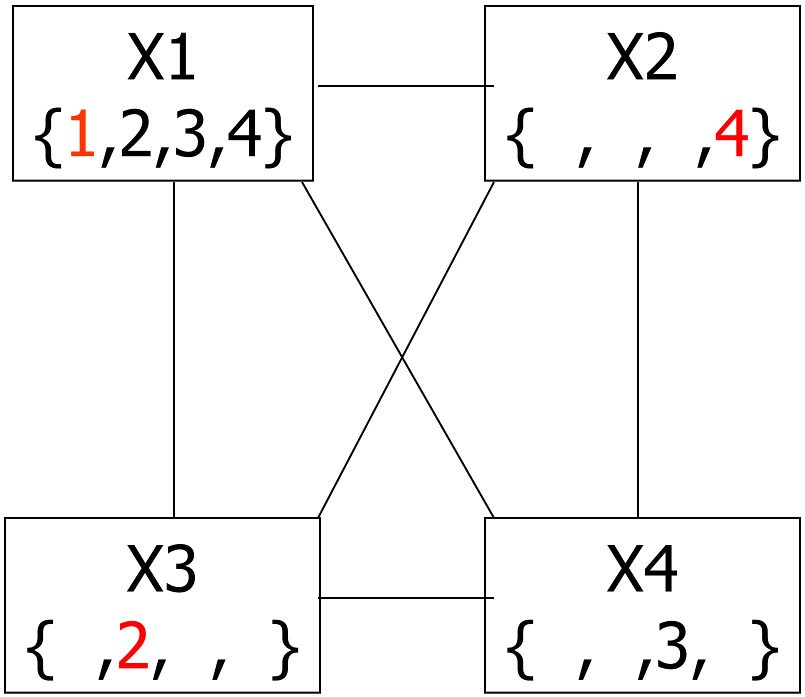
Example: 4-Queens Problem



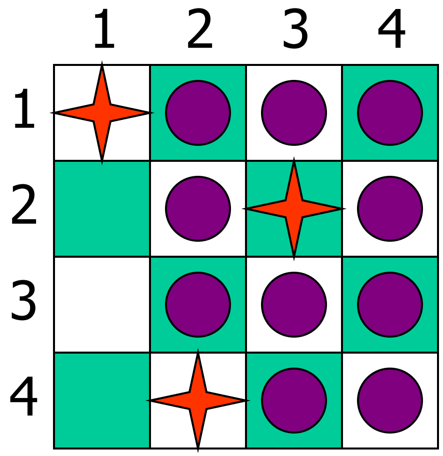


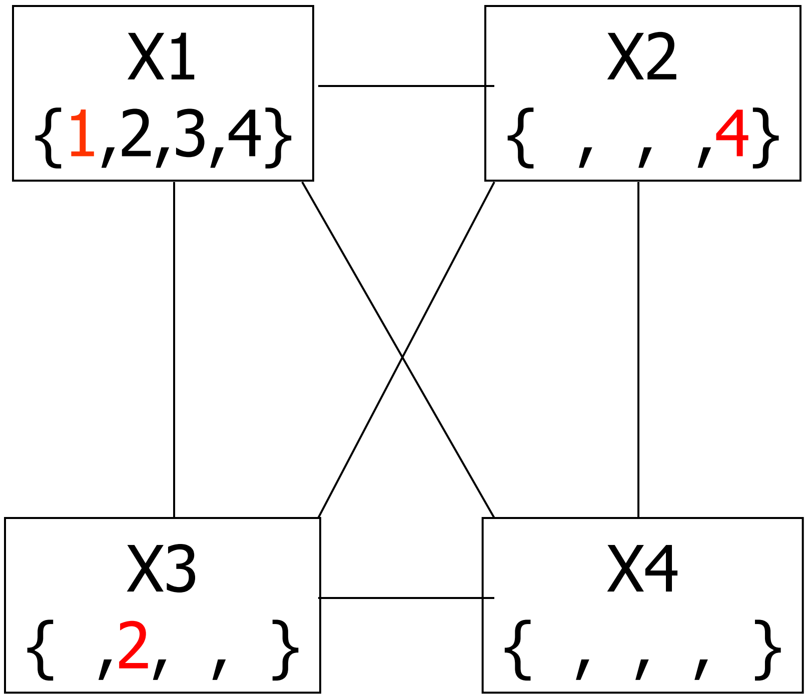
Example: 4-Queens Problem



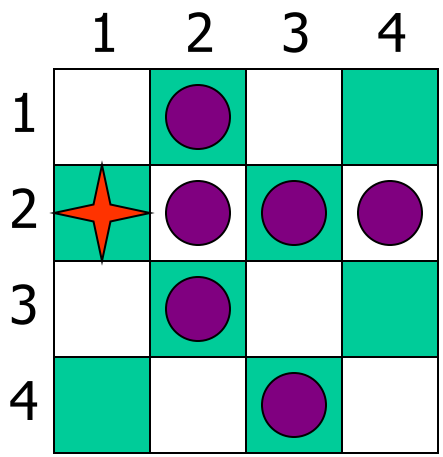


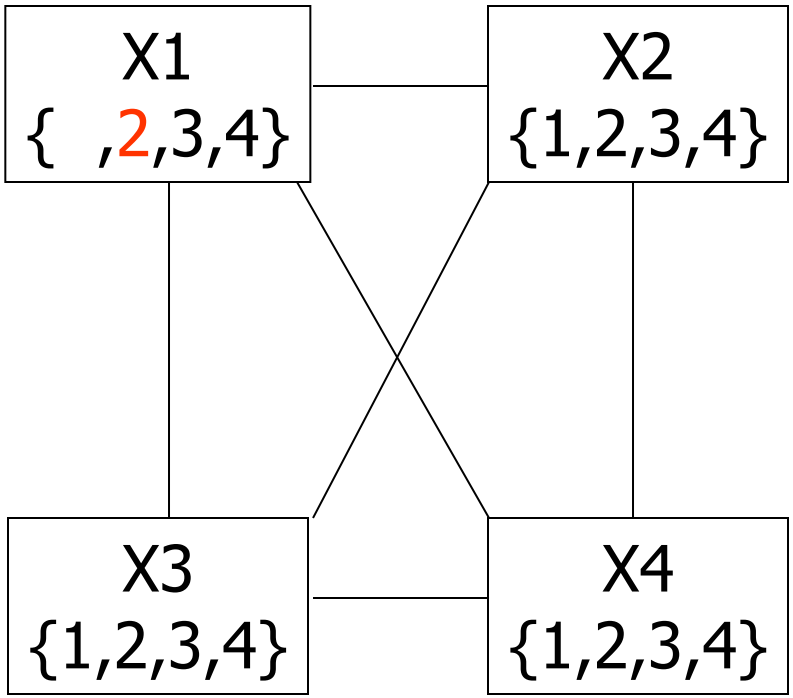
Example: 4-Queens Problem



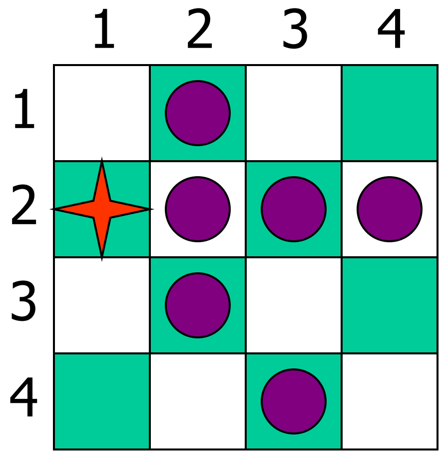


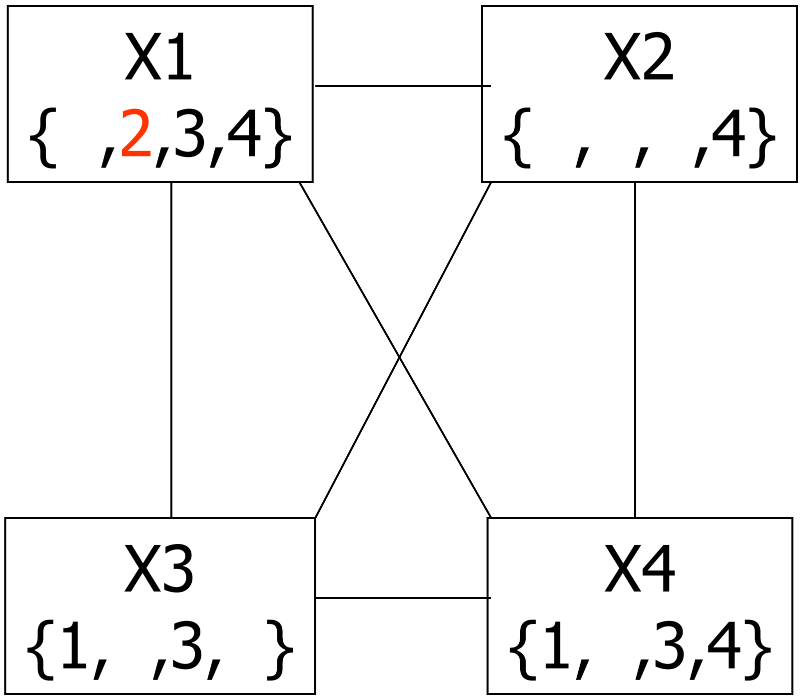
Example: 4-Queens Problem



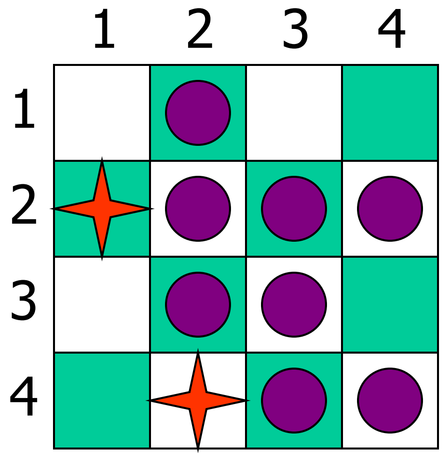


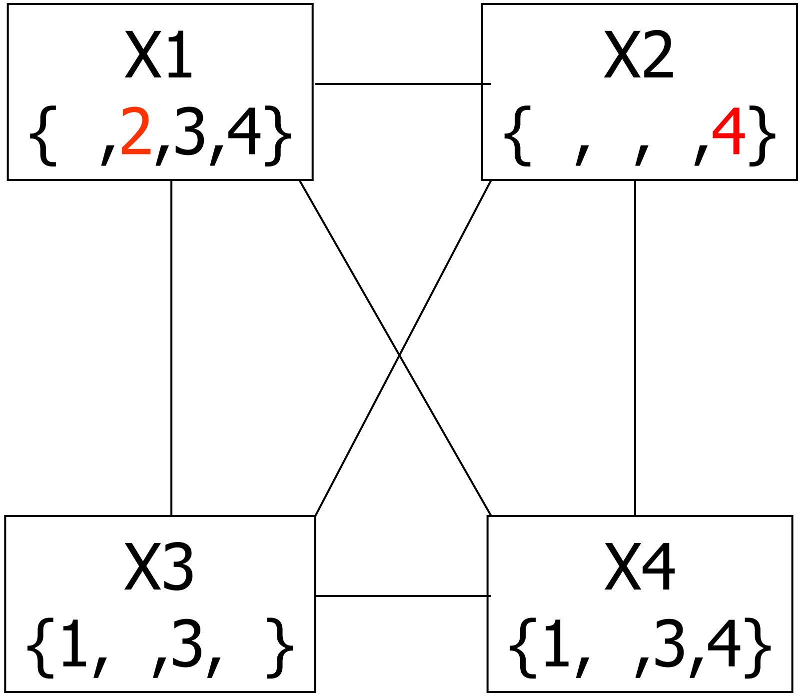
Example: 4-Queens Problem



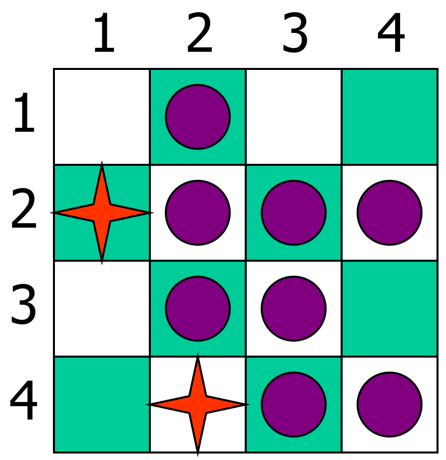


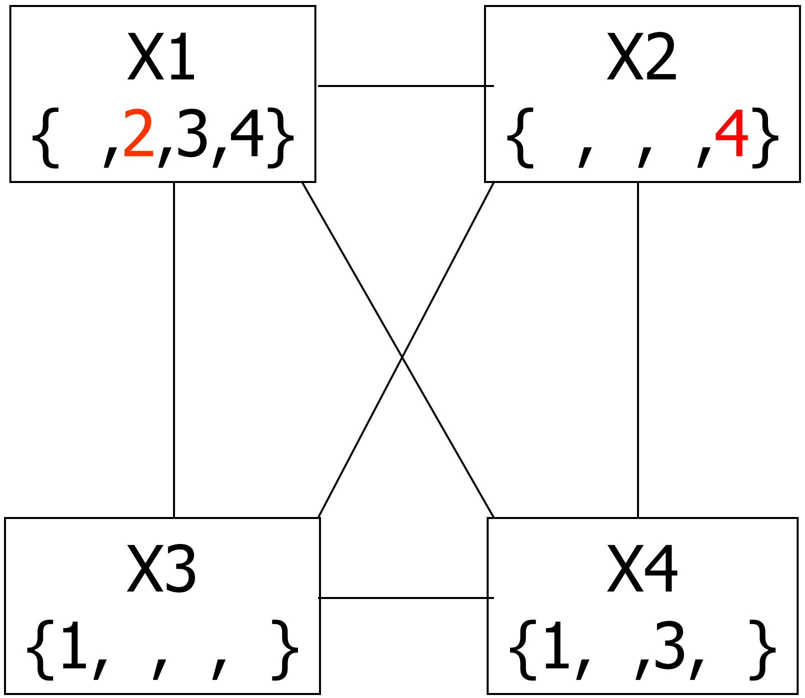
Example: 4-Queens Problem



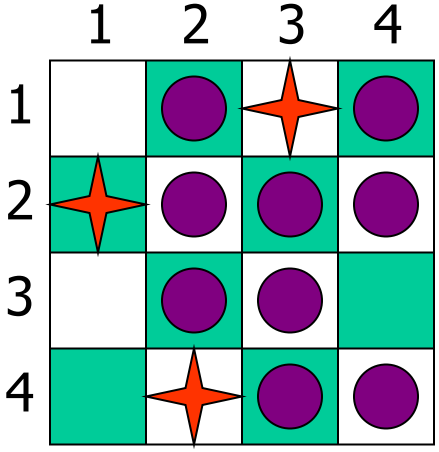


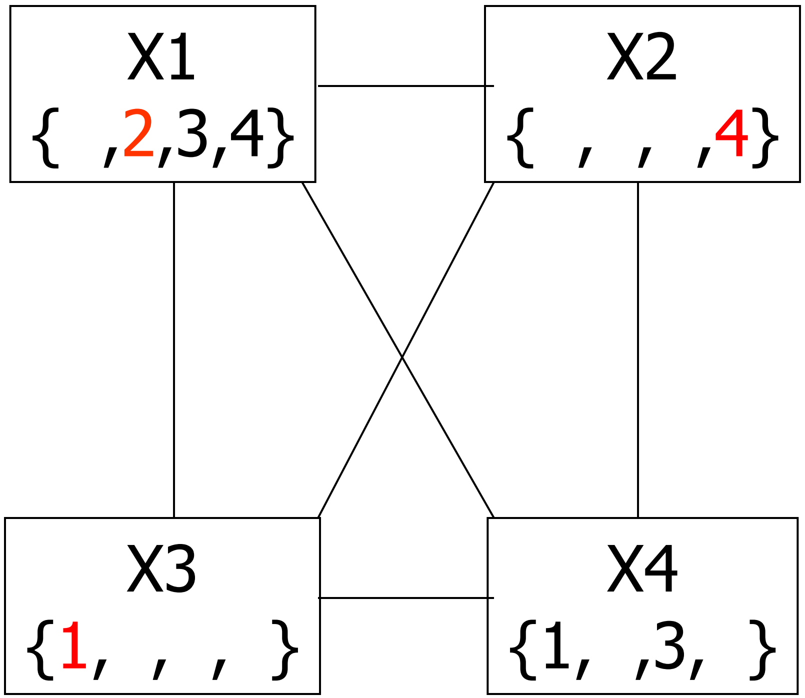
Example: 4-Queens Problem



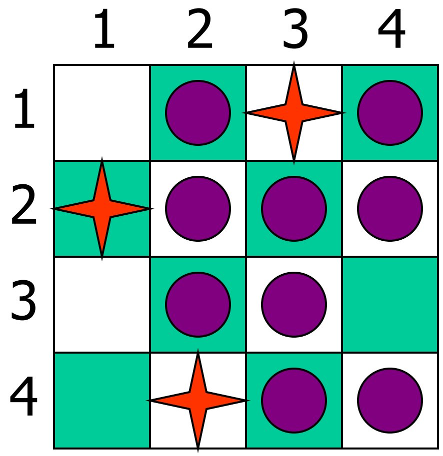


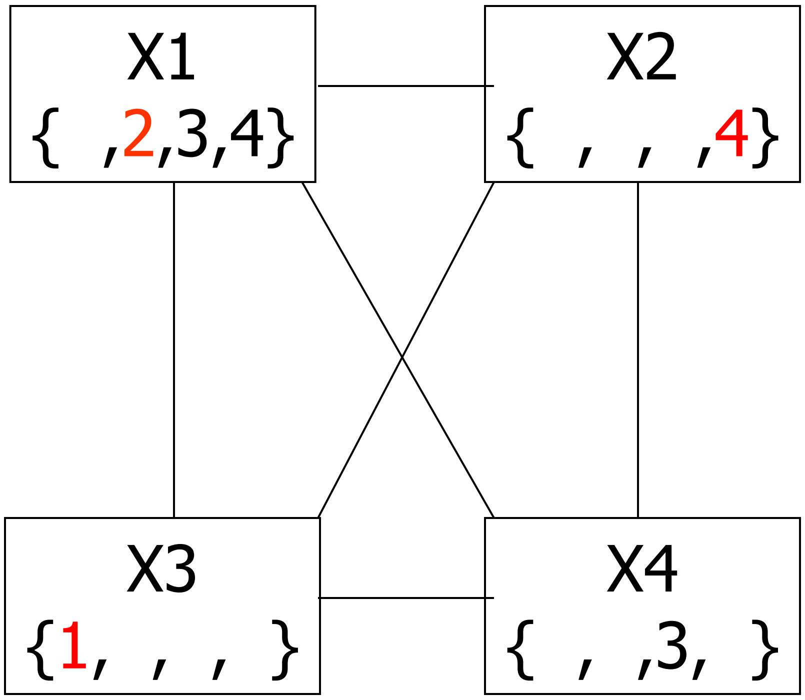
Example: 4-Queens Problem



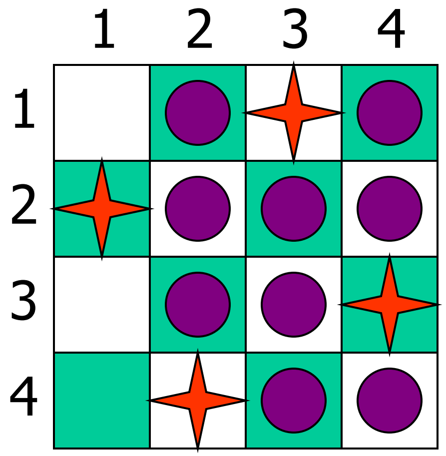


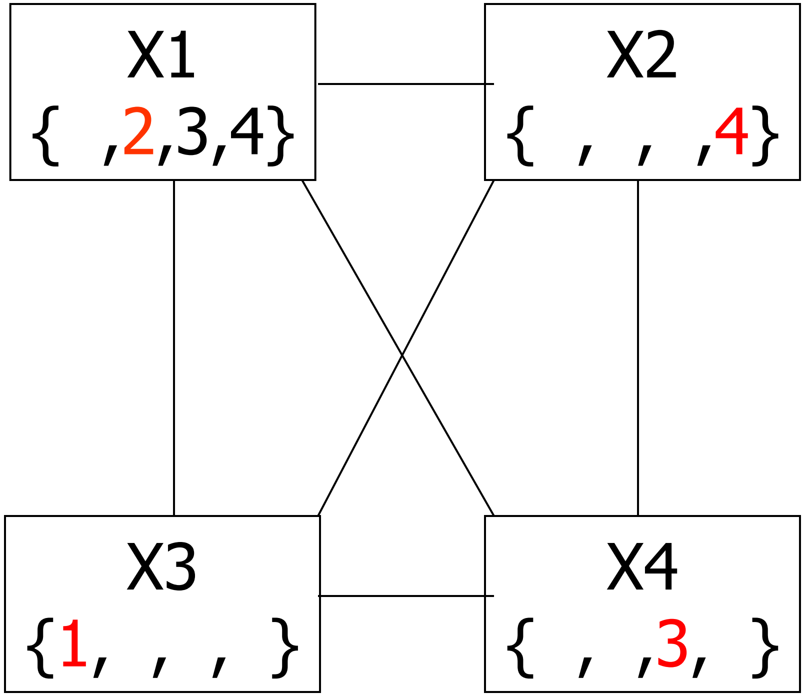
Example: 4-Queens Problem





Example: 4-Queens Problem

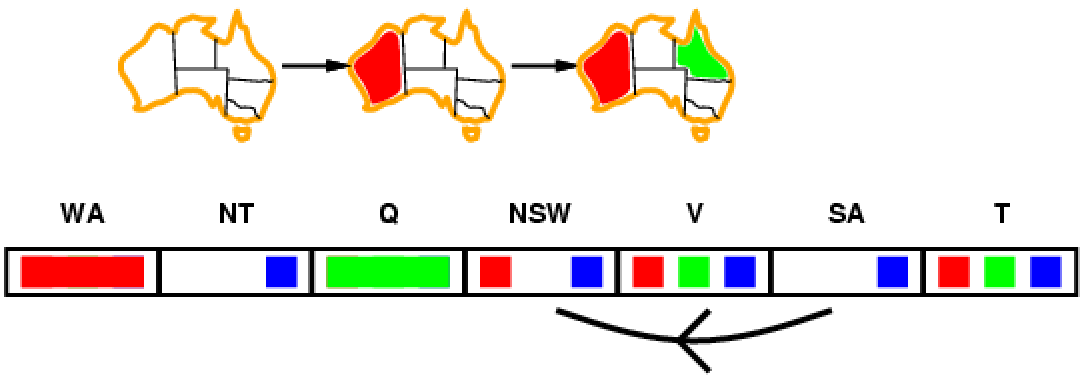




Constraint Propagation

- **Simplest form of propagation makes each arc consistent**
- ***Arc X →Y* (link in constraint graph) is consistent iff**
-

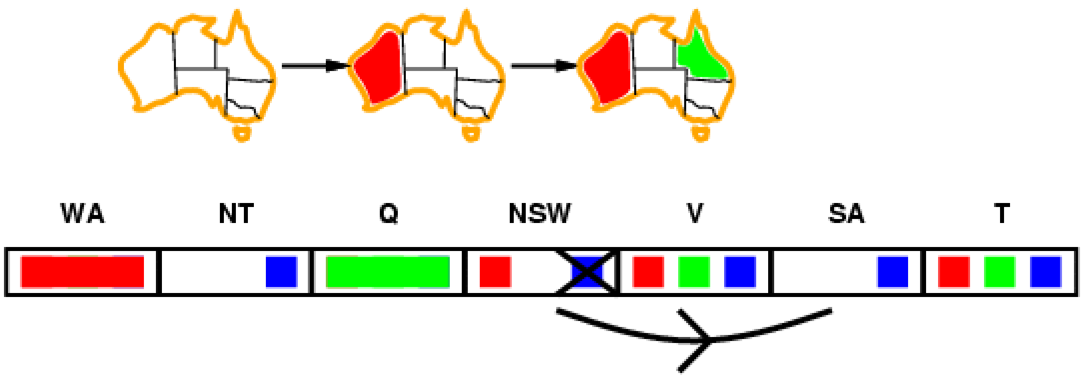
    for every value *x* of *X* there is some allowed *y*



Arc consistency

- **Simplest form of propagation makes each arc** **consistent**
- ***X* → *Y* is consistent iff**
-

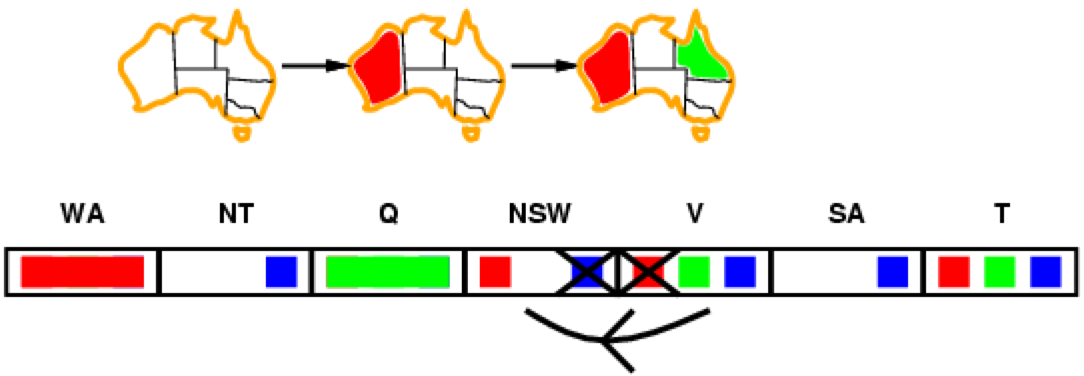
  for every value *x* of *X* there is some allowed *y*



Arc consistency

- **Simplest form of propagation makes each arc consistent**
- ***X* → *Y* is consistent iff**
-

for every value *x* of *X* there is some allowed *y*

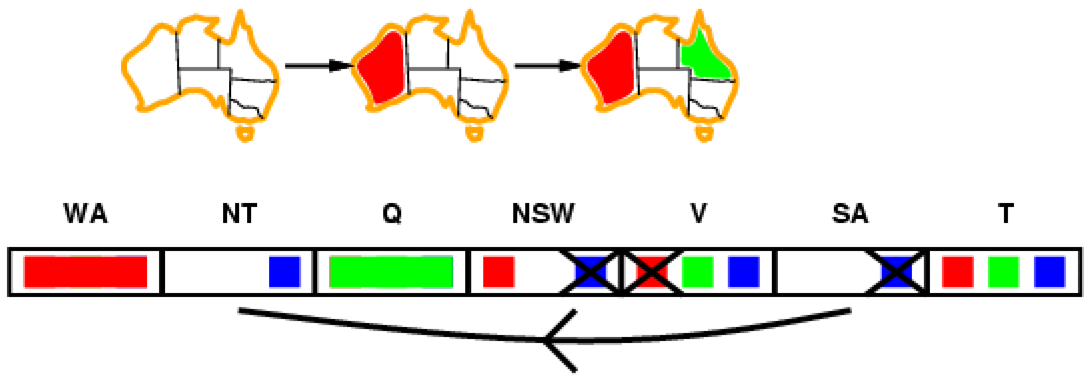


- **If *X* loses a value, neighbors of *X* need to be rechecked**

Arc consistency

- **Simplest form of propagation makes each arc consistent**
- ***X* → *Y* is consistent iff**
-

  for every value *x* of *X* there is some allowed *y*



- **If *X* loses a value, neighbors of *X* need to be rechecked**
- **Arc consistency detects failure earlier than forward checking**
- **Can be run as a preprocessor or after each assignment**

Arc Consistency Algorithm AC-3

function **AC-3(*csp*) %** returns the CSP, possibly with reduced domains

inputs**: *csp*, a binary csp with variables** $\{X_1, X_2, \ldots, X_n\}$

local variables: ***queue*, a queue of arcs initially the arcs in *csp***

while **queue is not empty** do

(X_i, X_j) ← **REMOVE-FIRST(*queue*)**

if **REMOVE-INCONSISTENT-VALUES(X_i, X_j)** then

for each X_k in **NEIGHBORS[X_i]** do

add (X_k, X_i) **to queue**

function **REMOVE-INCONSISTENT-VALUES(X_i, X_j) %** returns *true* iff a value is removed

removed* ← *false

for each ***x*** in **DOMAIN[X_i]** do

if **no value *y* in DOMAIN[X_j] allows (x,y) to satisfy the constraints between X_i and X_j**

then delete **x from DOMAIN[X_i]; *removed* ← *true***

return ***removed***

Time complexity: $O(n^2d^3)$

Local Search for CSPs

- **Hill-climbing methods typically work with "complete" states, i.e., all variables assigned**

- **To apply to CSPs:**

  - allow states with unsatisfied constraints
  - operators reassign variable values

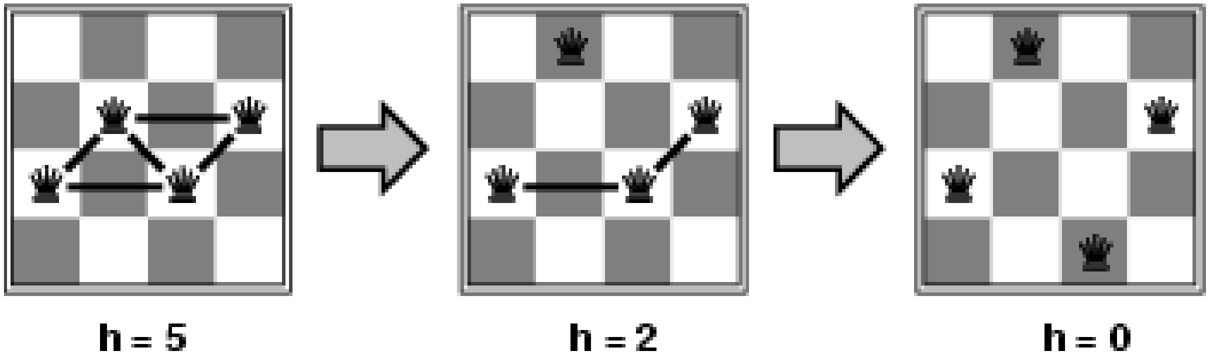
- **Variable selection: randomly select any conflicted variable**

- **Value selection by min-conflicts heuristic:**

  - choose value that violates the fewest constraints
  - i.e., hill-climb with *h(n)* = number of violated constraints

Example: n-queens

- **States**: **4 queens in 4 columns (4^4 = 256 states)**
- **Actions**: **move queen in column**
- **Goal test**: **no attacks**
- **Evaluation**: ***h(n)*** **= number of attacks**



- **Given random initial state, we can solve *n*-queens for large *n* with high probability**

Real-world CSPs

- **Assignment problems**
  - e.g., who teaches what class
- **Timetabling problems**
  - e.g., which class is offered when and where?
- **Transportation scheduling**
- **Factory scheduling**
- **Notice that many real-world problems involve real-valued variables**

Summary

- **CSPs are a special kind of problem:**
  - states defined by values of a fixed set of variables
  - goal test defined by constraints on variable values
- **Backtracking = depth-first search with one variable assigned per node**
- **Variable ordering and value selection heuristics help significantly**
- **Forward checking prevents assignments that guarantee later failure**
- **Constraint propagation (e.g., arc consistency) additionally constrains values and detects inconsistencies**
- **Iterative min-conflicts is usually effective in practice**