ARTIFICIAL INTELLIGENCE

Russell & Norvig

Chapter 6. Constraint Satisfaction Problems

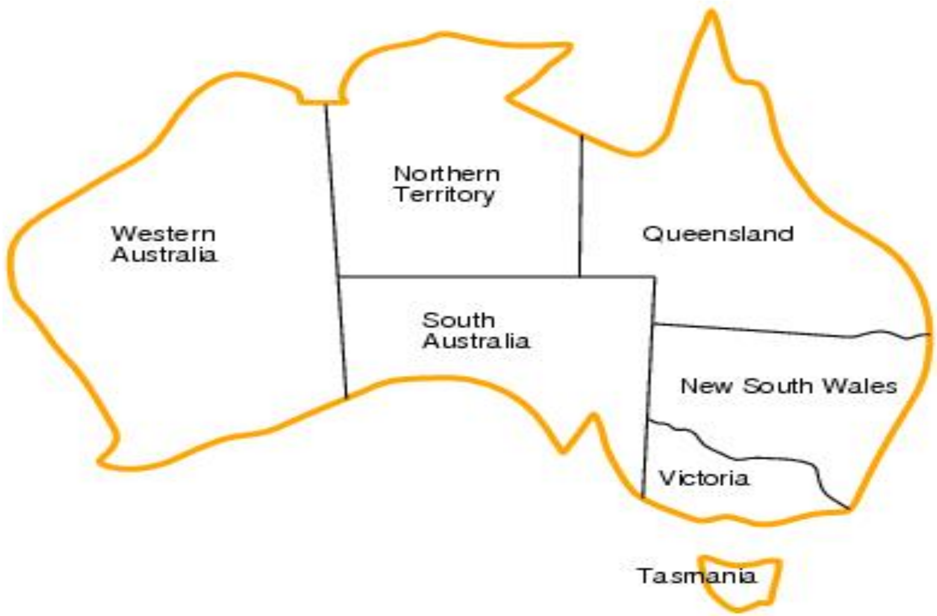
Constraint Satisfaction Problems

- 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 defined as an *assignment* of values to some or all variables.
- *Consistent assignment*
  - assignment does not violate the constraints
- CSP benefits
  - Standard representation pattern
  - Generic goal and successor functions
  - Generic heuristics (no domain specific expertise).

CSPs (continued)

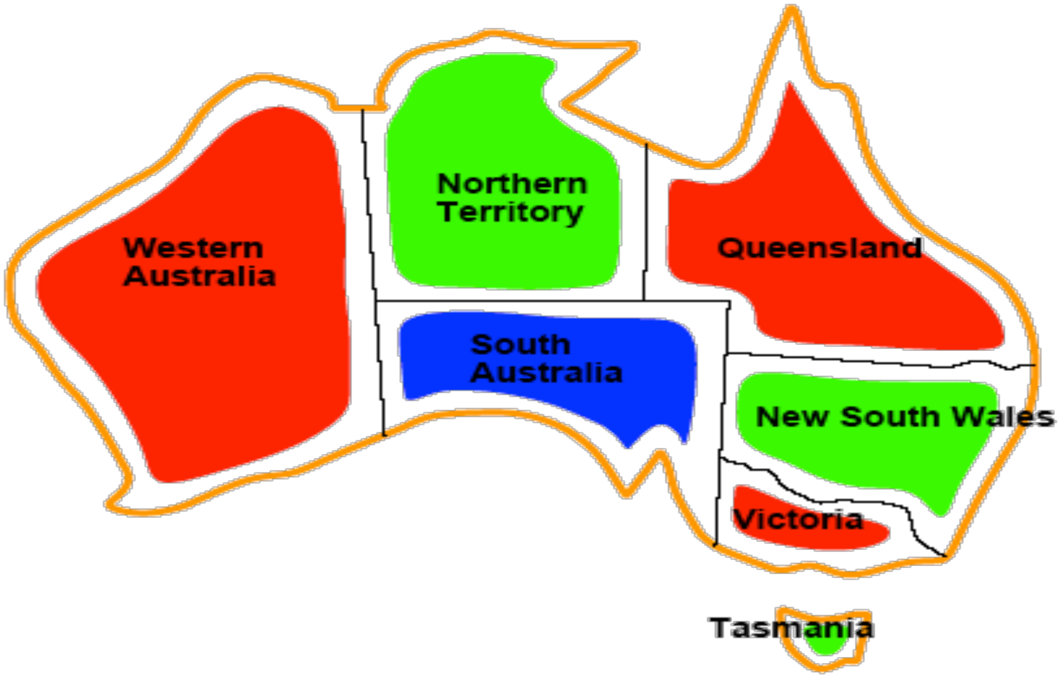
- An assignment is *complete* when every variable is mentioned.
- A *solution* to a CSP is a complete assignment that satisfies all constraints.
- Some CSPs require a solution that maximizes an *objective function*.
- Examples of Applications:
  - Scheduling of rooms, airline schedules, etc
  - Cryptography
  - Sudoku and lots of other puzzles
  - Registering for classes

CSP 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. $WA \neq NT$

CSP example: map coloring



- Solutions are assignments satisfying all constraints, e.g.
  {WA=red,NT=green,Q=red,NSW=green,V=red,SA=blue,T=green}

Graph coloring

- More general problem than map coloring

- Planar graph = graph in the 2d-plane with no edge crossings

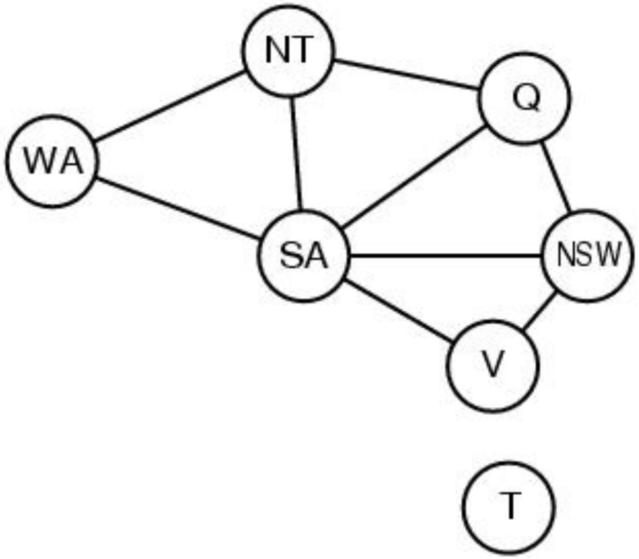
- Guthrie's conjecture (1852)

  Every planar graph can be colored with 4 colors or less

  - Proved (using a computer) in 1977 (Appel and Haken)

Constraint graphs

- Constraint graph:
  - nodes are variables
  - arcs are binary constraints



- Graph can be used to simplify search
  e.g. Tasmania is an independent subproblem

(will return to graph structure later)

Varieties of CSPs

- Discrete variables
  - Finite domains; size $d \Rightarrow O(d^n)$ complete assignments.
    - E.g. Boolean CSPs: 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 $StartJob_1 + 5 \leq StartJob_3$.
    - Infinitely many solutions
    - Linear constraints: solvable
    - Nonlinear: no general algorithm
- Continuous variables
  - e.g. building an airline schedule or class schedule.
  - Linear constraints solvable in polynomial time by LP methods.

Varieties of constraints

- Unary constraints involve a single variable.
  - e.g. $SA \neq green$
- Binary constraints involve pairs of variables.
  - e.g. $SA \neq WA$
- Higher-order constraints involve 3 or more variables.
  - Professors A, B,and C cannot be on a committee together
  - Can always be represented by multiple binary constraints
- Preference (soft constraints)
  - e.g. *red* is better than *green* often can be represented by a cost for each variable assignment
  - combination of optimization with CSPs

CSP as a standard search problem

- A CSP can easily be expressed as a standard search problem.
- Incremental formulation
  - *Initial State*: the empty assignment {}
  - *Successor function*: Assign a value to any unassigned variable provided that it does not violate a constraint
  - *Goal test*: the current assignment is complete and consistent
  - *Path cost*: constant cost for every step (not generally relevant)
- Can also use complete-state formulation
  - Local search techniques tend to work well

CSP as a standard search problem

- Solution is found at depth *n* (if there are *n* variables).

- Consider using BFS
  - Branching factor *b* at the top level is *nd*
  - At next level is (n-1)d
  - ….

- end up with $n!d^n$ leaves even though there are only d^n complete assignments!

Commutativity

- CSPs are commutative.
  - The order of any given set of actions has no effect on the outcome.
  - Example: choose colors for Australian territories one at a time
    - [WA=red then NT=green] same as [NT=green then WA=red]
- All CSP search algorithms can generate successors by considering assignments for only a single variable at each node in the search tree

  $\Rightarrow$ there are d^n leaves

  (will need to figure out later which variable to assign a value to at each node)

Backtracking search

- Similar to Depth-first search
- Chooses values for one variable at a time and backtracks when a variable has no legal values left to assign.
- Uninformed algorithm
  - Not good general performance

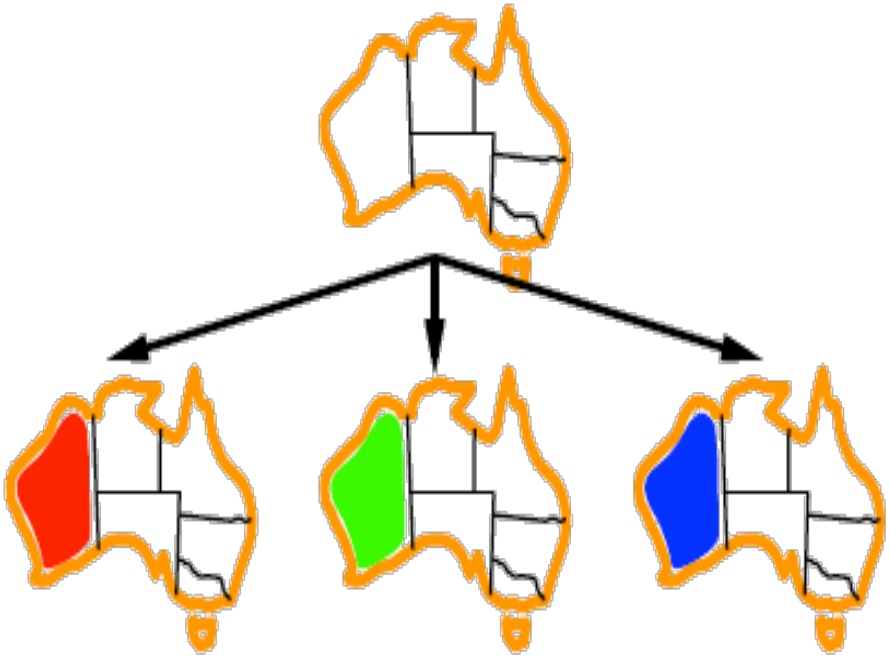
Backtracking search

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

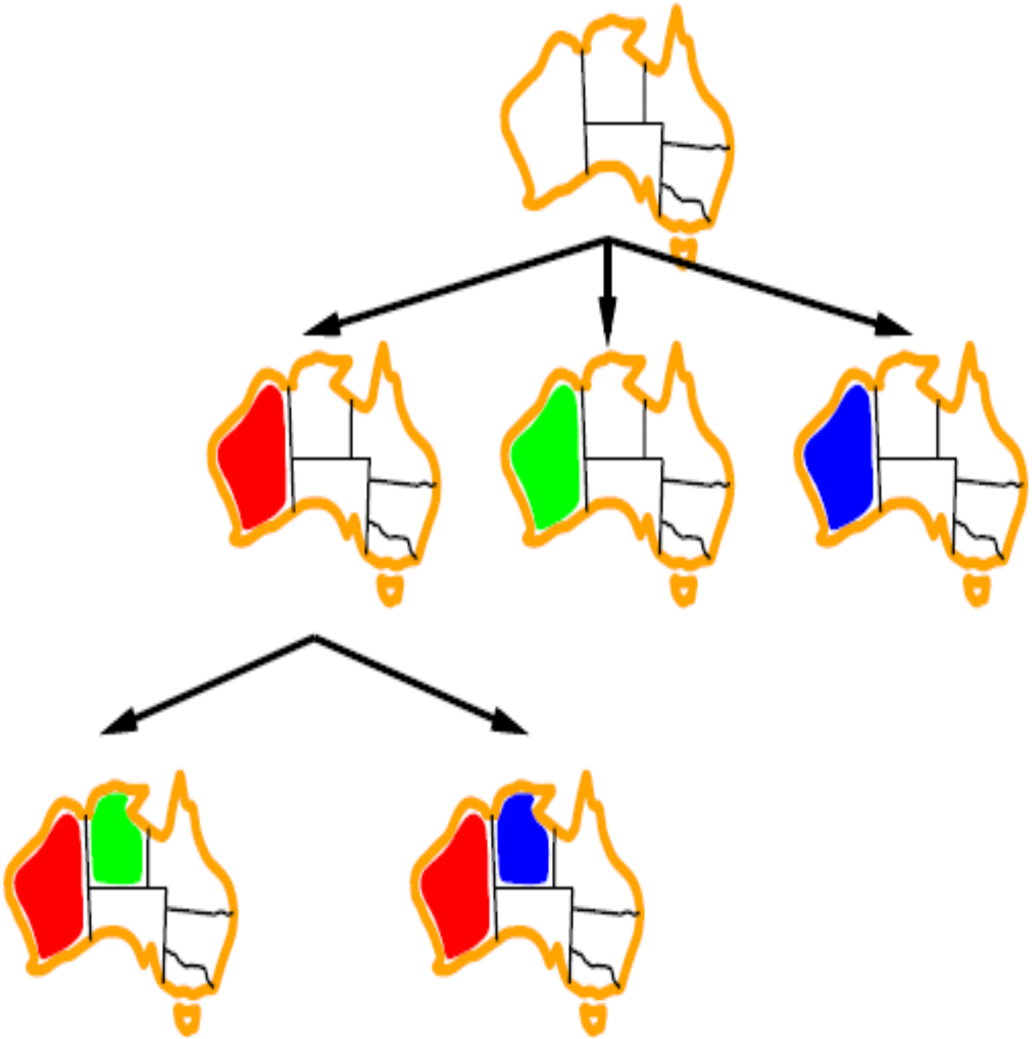
function RECURSIVE-BACKTRACKING(*assignment, csp*) **return** a solution or failure
  if *assignment* is complete **then return** *assignment*
  var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[*csp*],*assignment*,*csp*)
  for each *value* **in** ORDER-DOMAIN-VALUES(*var, assignment, csp*) **do**
      if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

Backtracking example

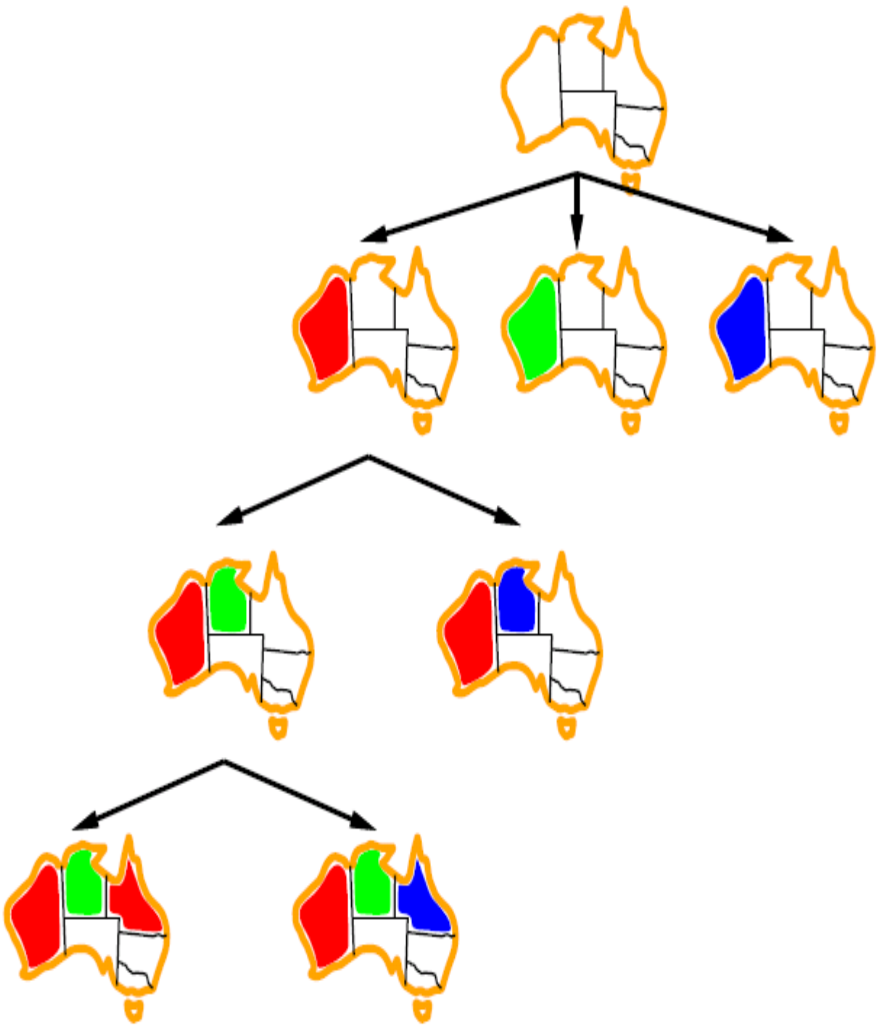
Backtracking example



Backtracking example



Backtracking example



Improving CSP efficiency

- Previous improvements on uninformed search
  → introduce heuristics

- For CSPs, general-purpose methods can give large gains in speed, e.g.,
  - Which variable should be assigned next?
  - In what order should its values be tried?
  - Can we detect inevitable failure early?
  - Can we take advantage of problem structure?

Note: CSPs are somewhat generic in their formulation, and so the heuristics are more general compared to methods considered earlier

Backtracking search

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

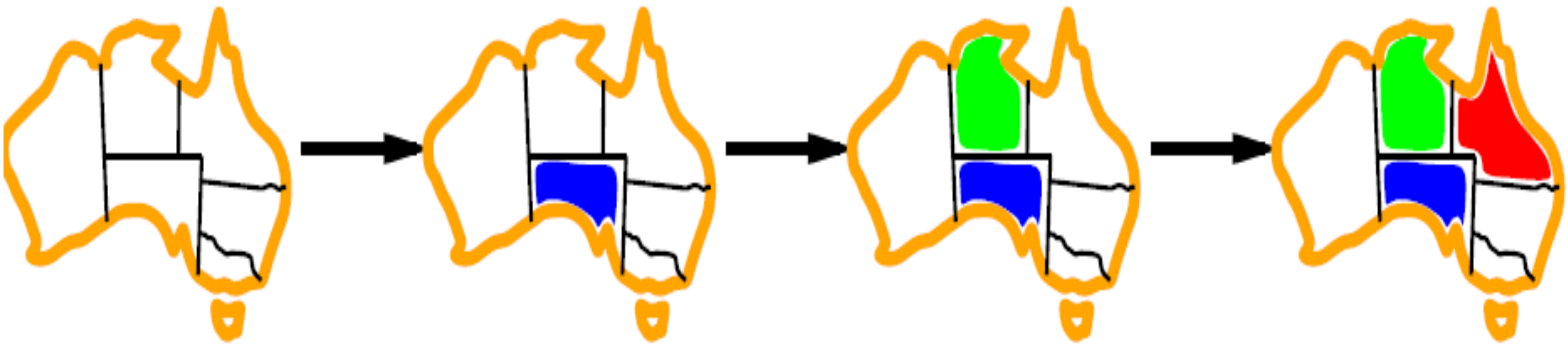
function RECURSIVE-BACKTRACKING(*assignment, csp*) **return** a solution or failure
    if *assignment* is complete **then return** *assignment*
    var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[*csp*],*assignment*,*csp*)
    for each *value* **in** ORDER-DOMAIN-VALUES(*var, assignment, csp*) **do**
  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

Minimum remaining values (MRV)

var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[*csp*],*assignment*,*csp*)

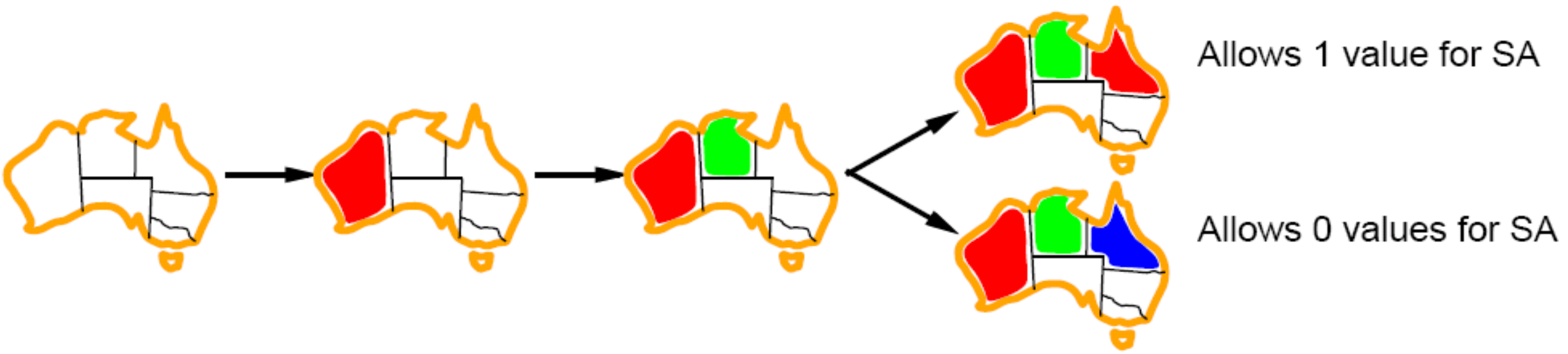
- A.k.a. most constrained variable heuristic
- *Heuristic Rule*: choose variable with the fewest legal moves
  - e.g., will immediately detect failure if X has no legal values

Degree heuristic for the initial variable



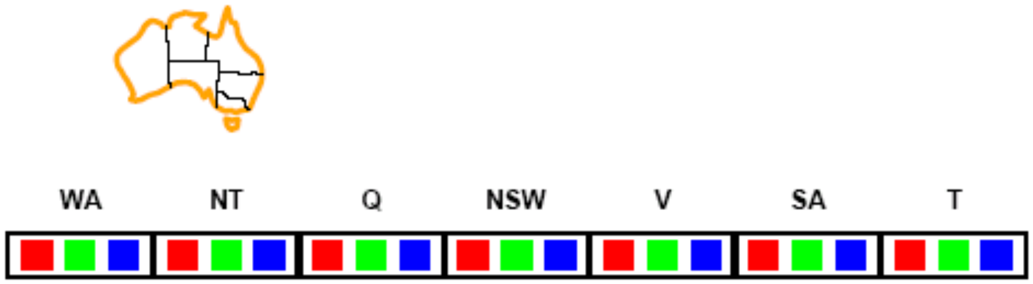
- *Heuristic Rule*: select variable that is involved in the largest number of constraints on other unassigned variables.
- Degree heuristic can be useful as a tie breaker.
- *In what order should a variable's values be tried?*

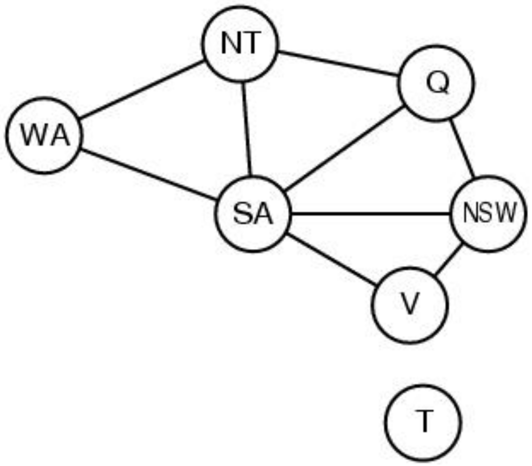
Least constraining value



- Least constraining value heuristic
- Used to select order of values
- Heuristic Rule: given a variable choose the least constraining value
  - leaves the maximum flexibility for subsequent variable assignments

Forward checking



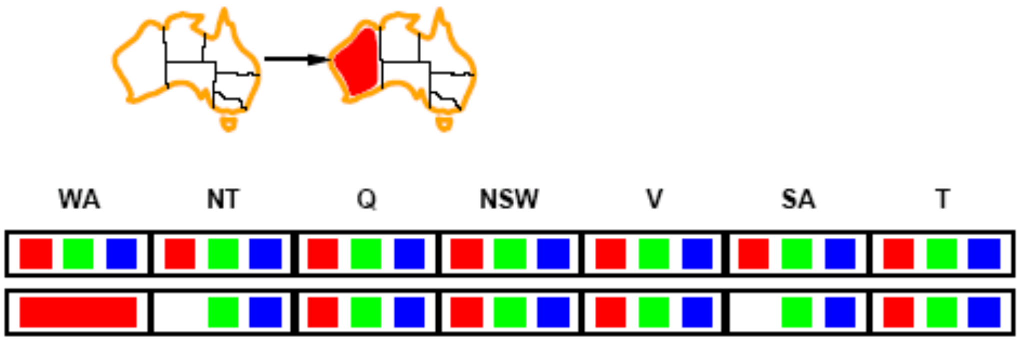


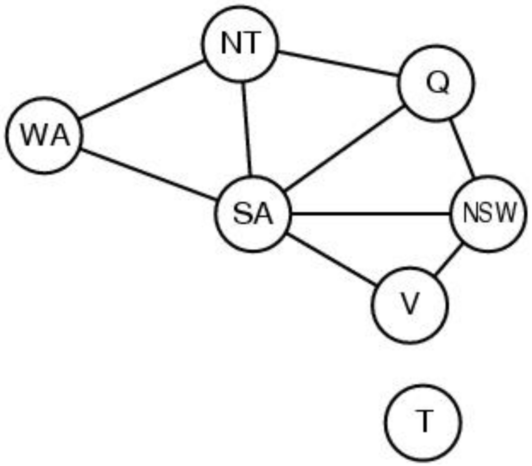
- Can we detect inevitable failure early?
  - *And avoid it later?*

- *Forward checking idea:* keep track of remaining legal values for unassigned variables.

- Terminate search when any variable has no legal values.

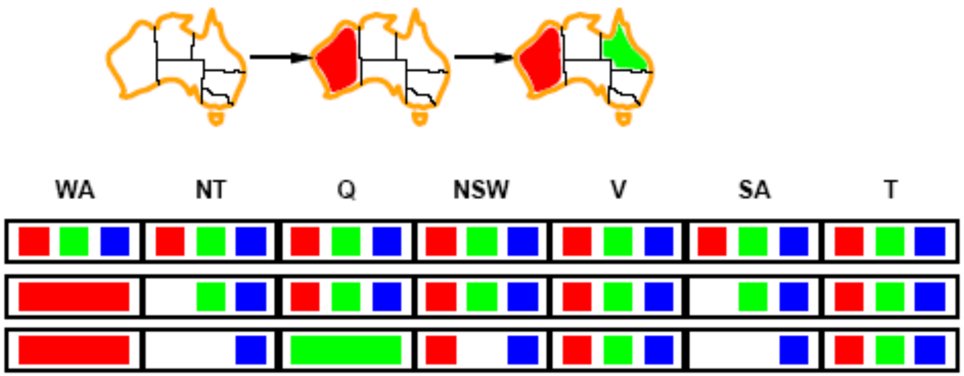
Forward checking

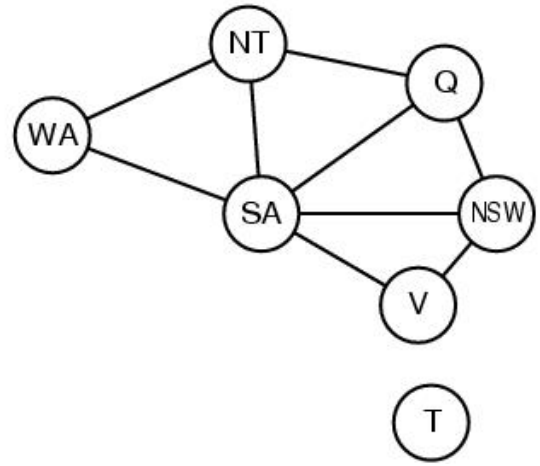




- Assign *{WA=red}*
- Effects on other variables connected by constraints to WA
  - *NT can no longer be red*
  - *SA can no longer be red*

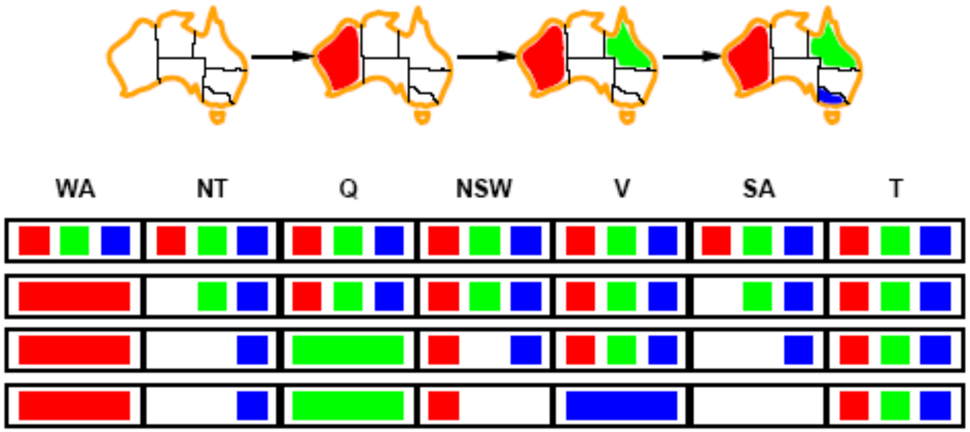
Forward checking

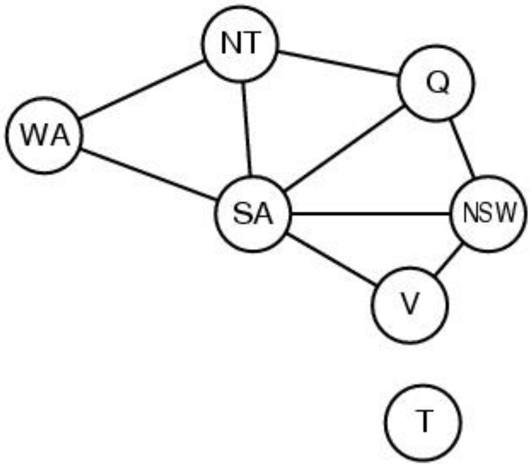




- Assign *{Q=green}*
- Effects on other variables connected by constraints with WA
  - *NT can no longer be green*
  - *NSW can no longer be green*
  - *SA can no longer be green*
- *MRV (minimum remaining values) heuristic* would automatically select NT or SA next

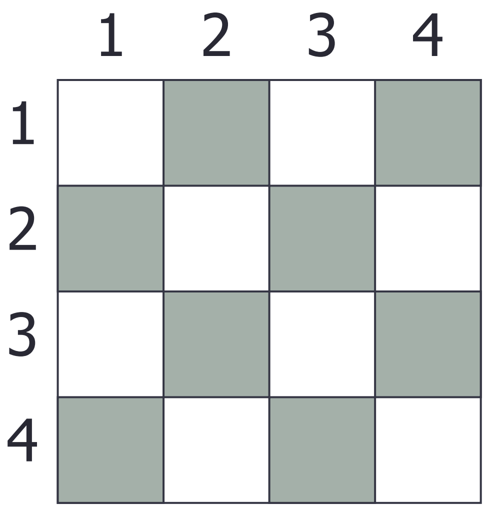
Forward checking

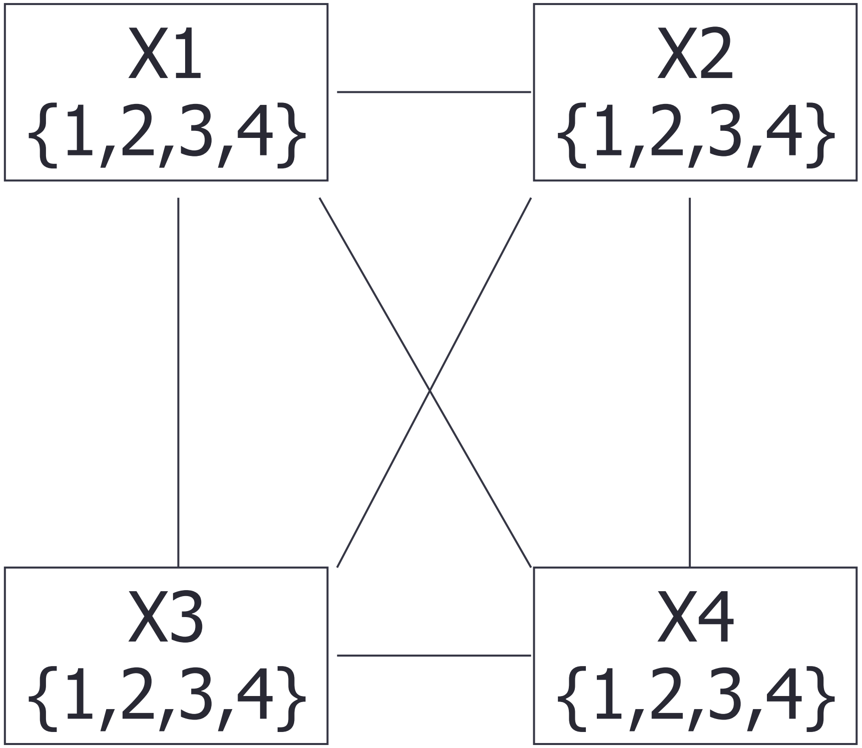




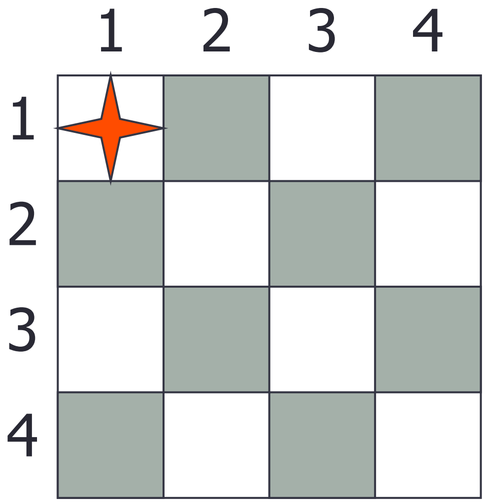
- If *V* is assigned *blue*
- Effects on other variables connected by constraints with WA
  - *NSW can no longer be blue*
  - *SA is empty*
- FC has detected that partial assignment is *inconsistent* with the constraints and backtracking can occur.

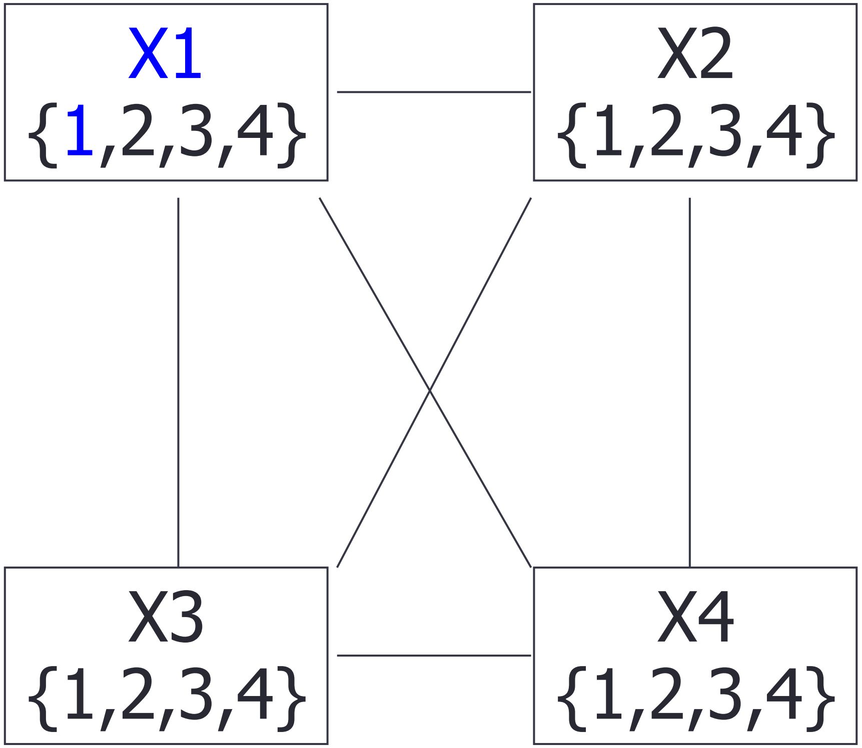
Example: 4-Queens Problem



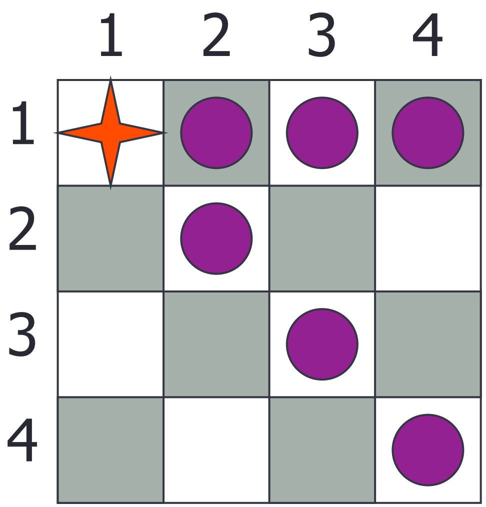


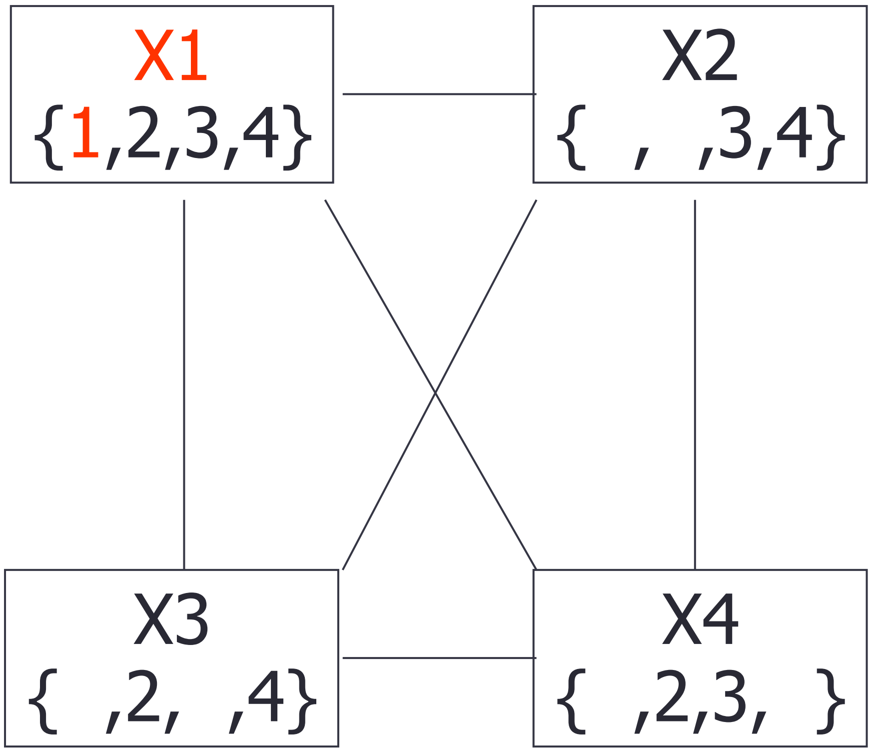
Example: 4-Queens Problem



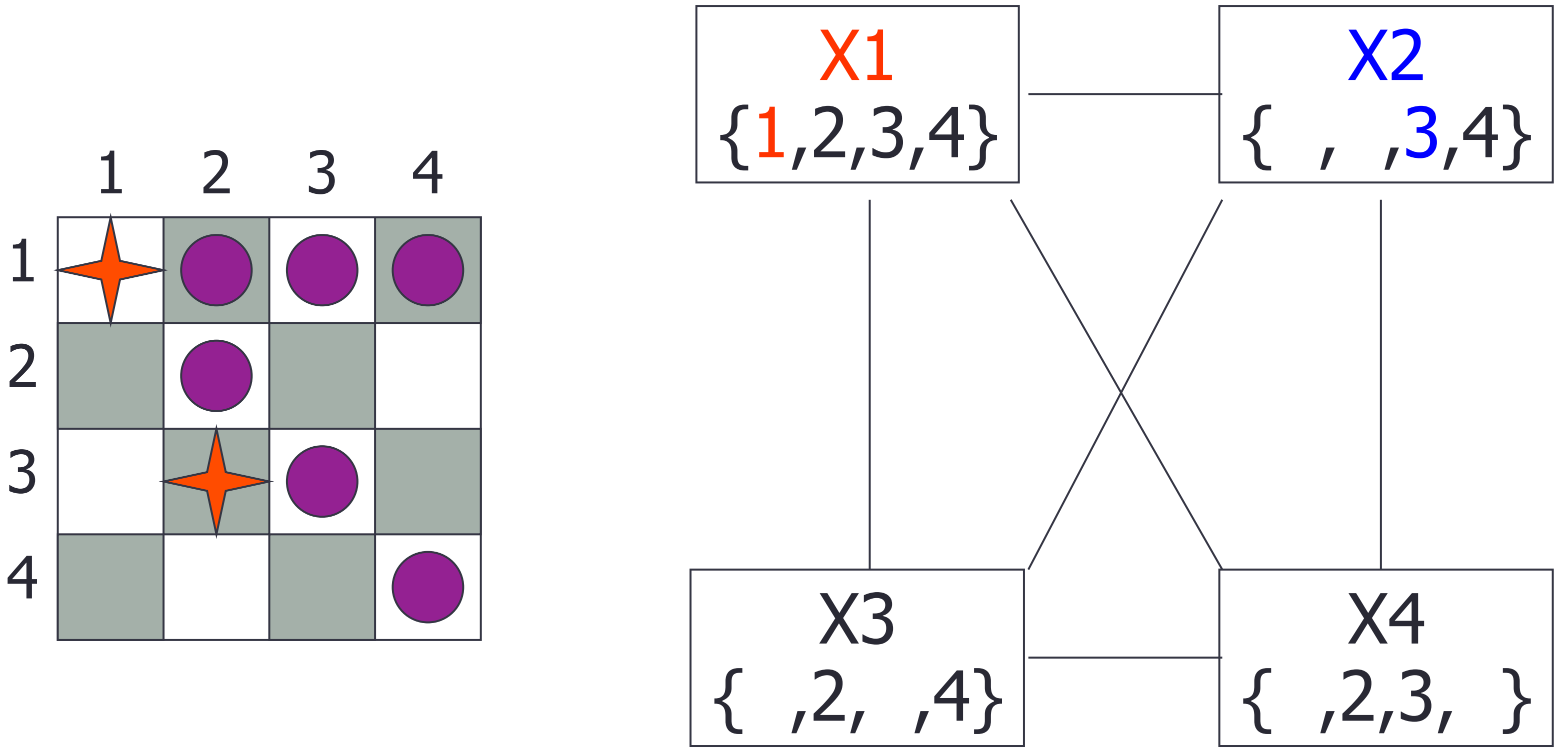


Example: 4-Queens Problem

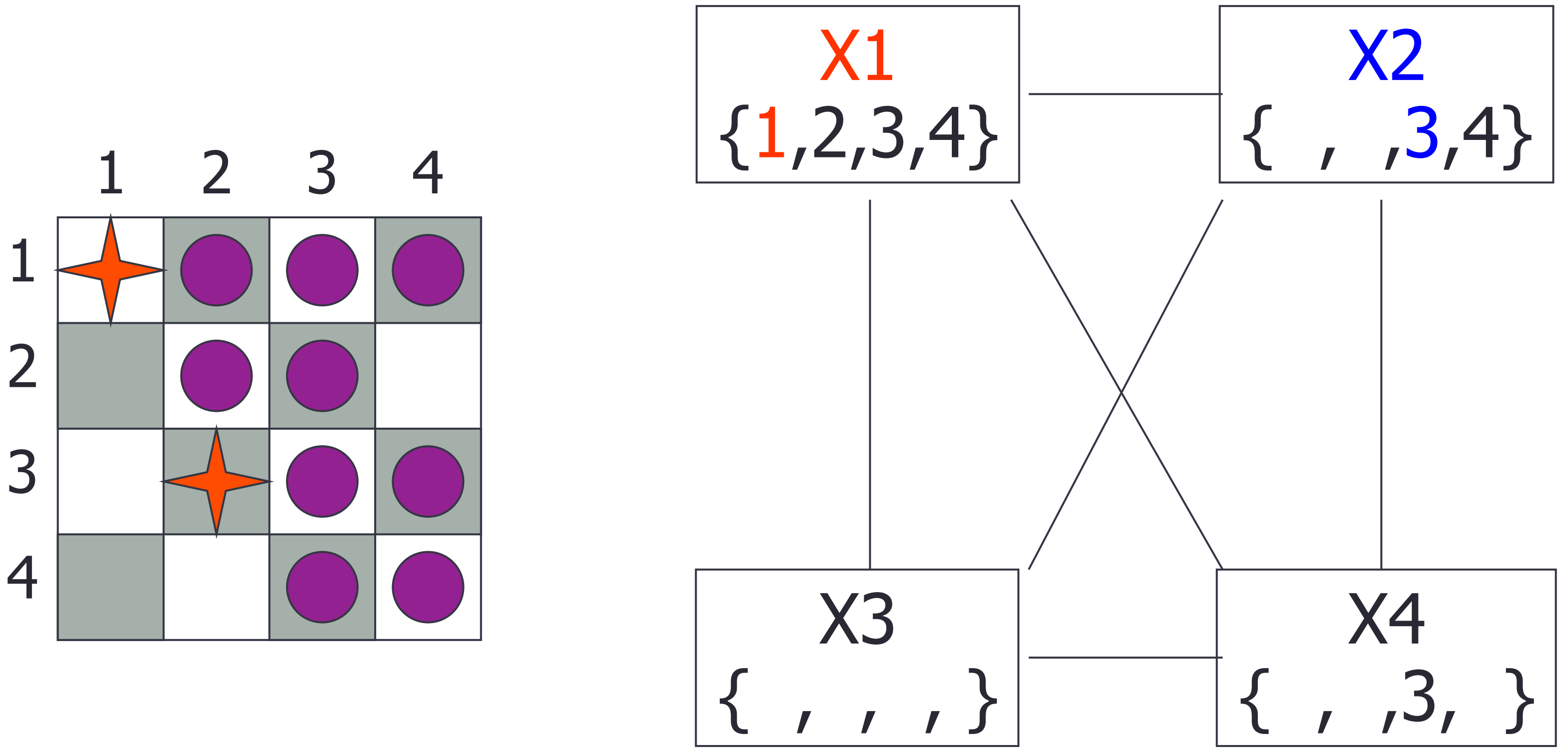




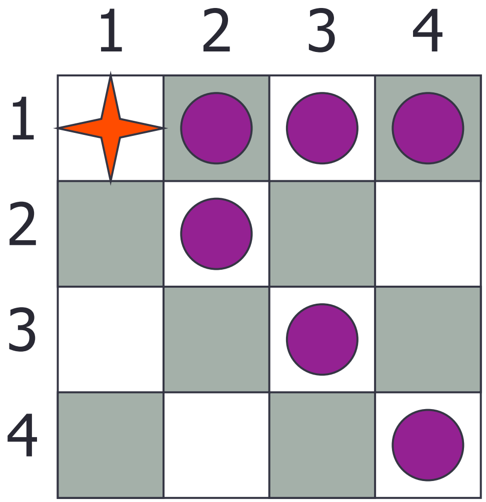
Example: 4-Queens Problem

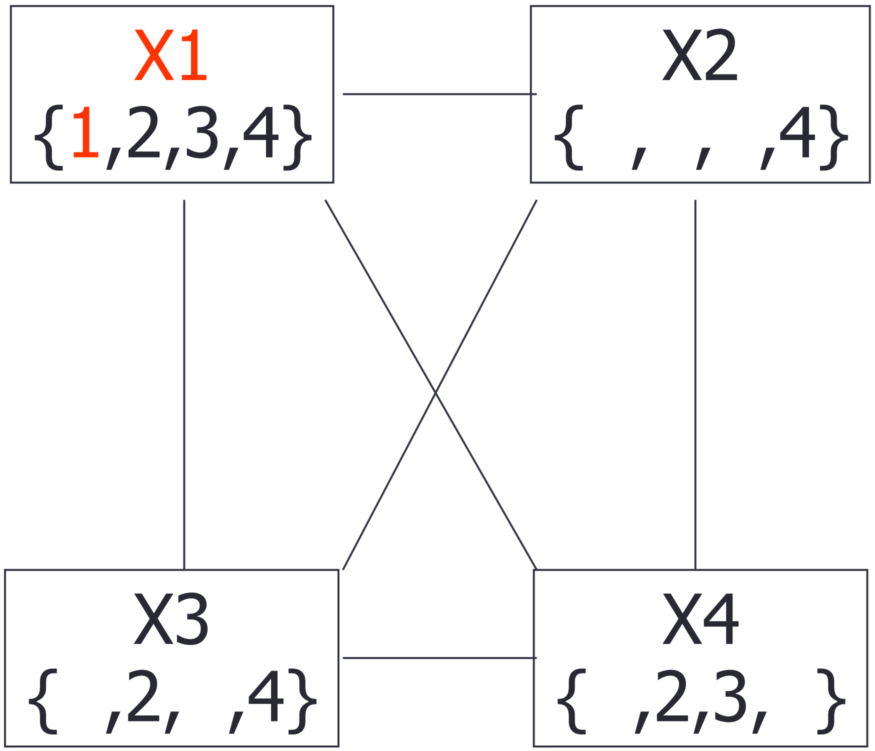


Example: 4-Queens Problem

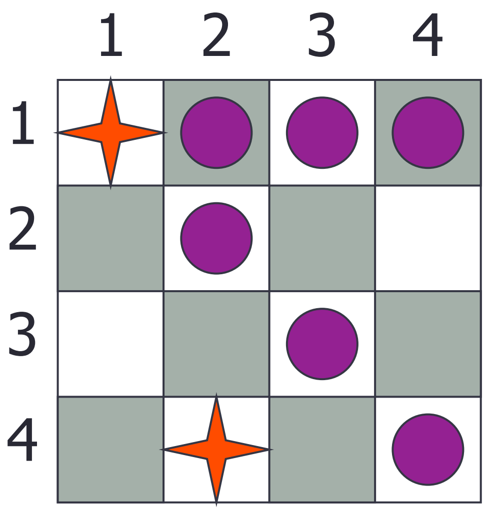


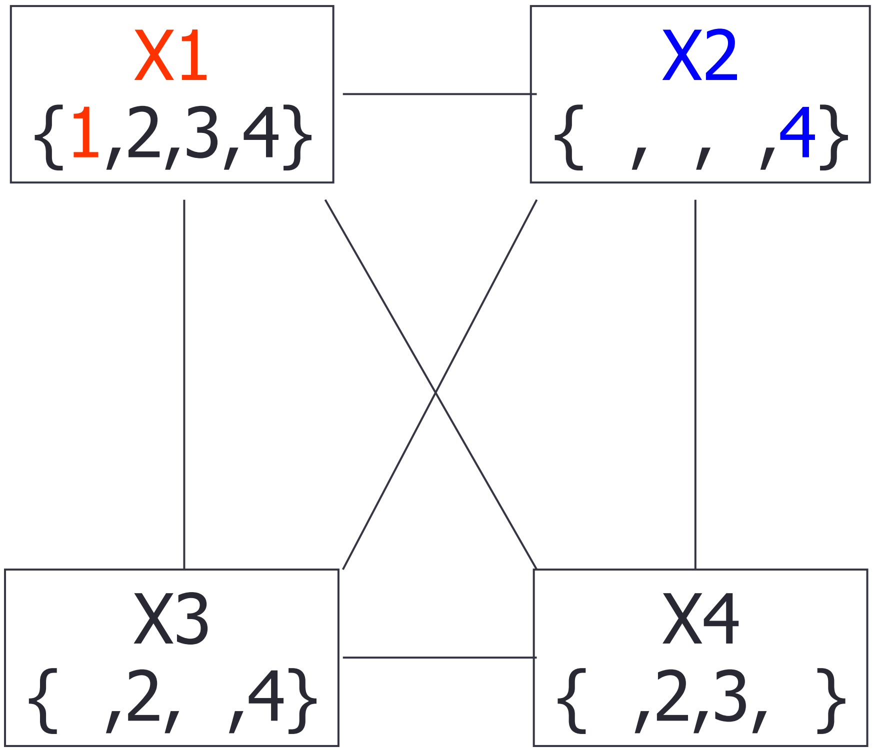
Example: 4-Queens Problem



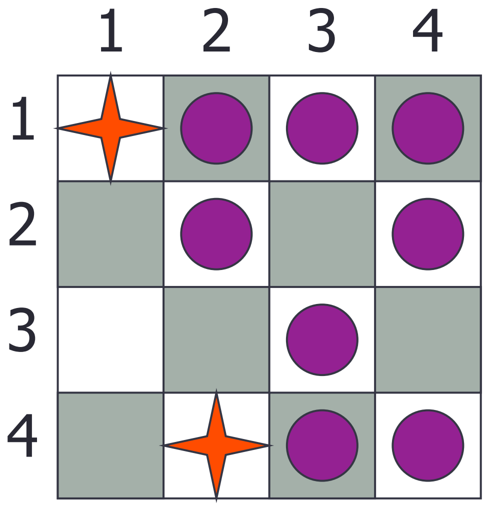


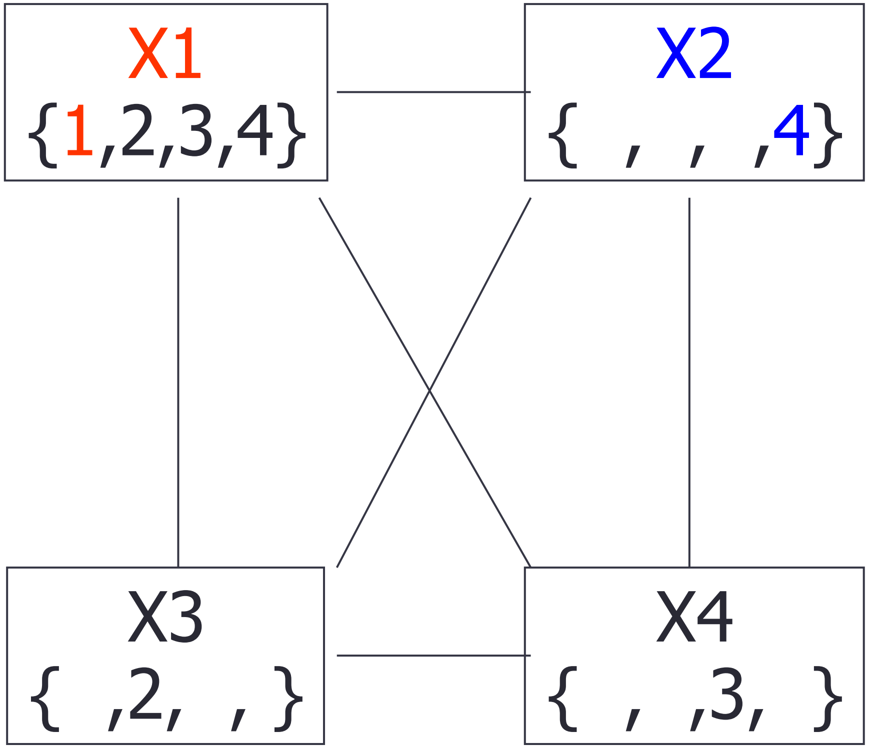
Example: 4-Queens Problem



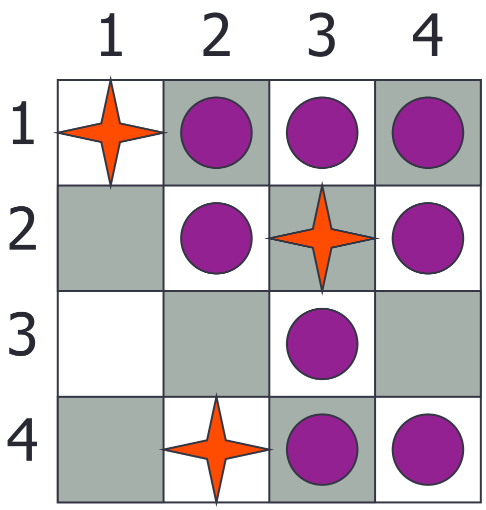


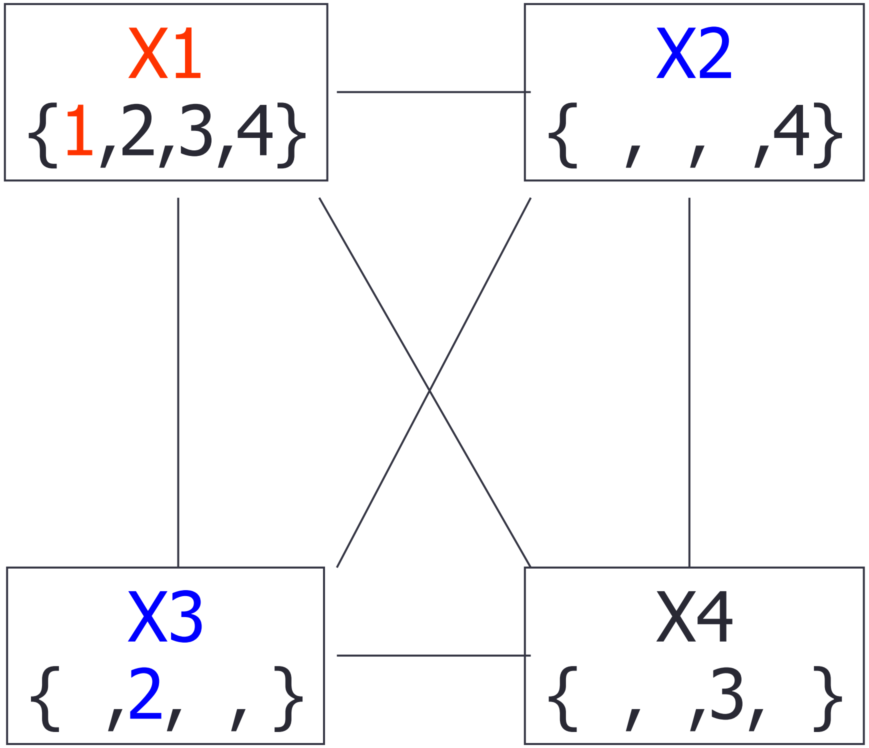
Example: 4-Queens Problem



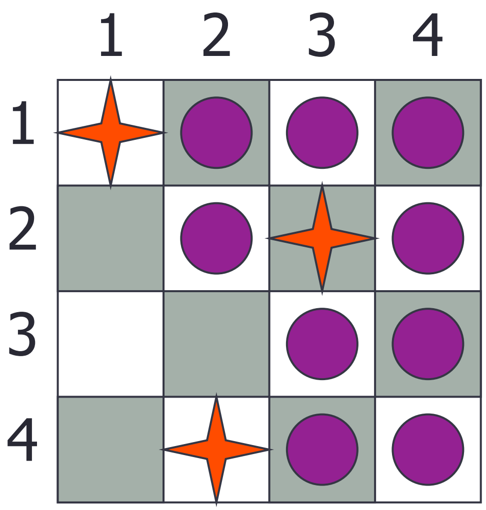


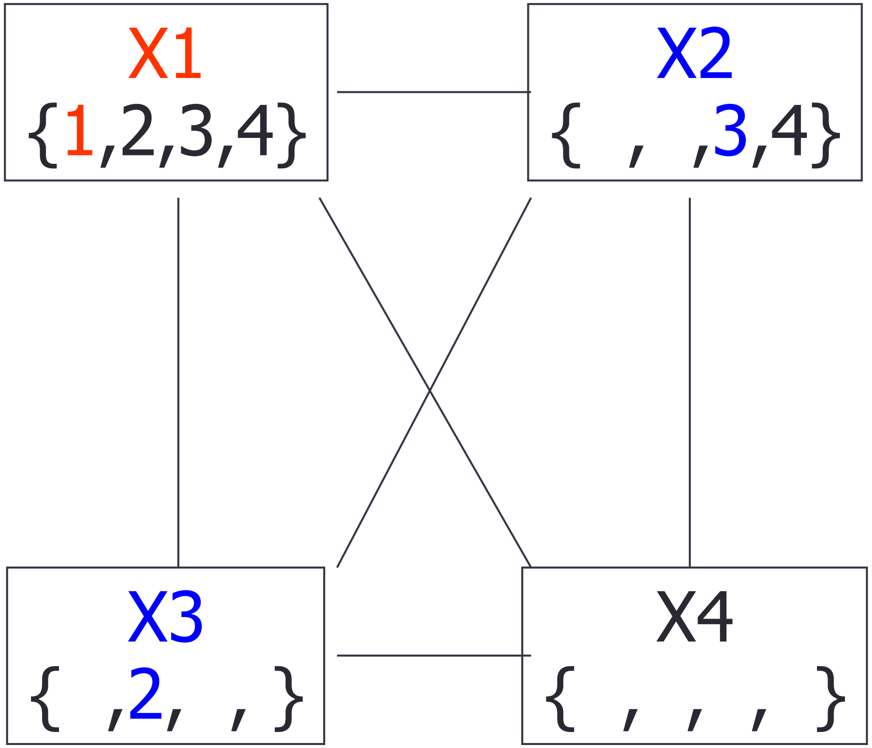
Example: 4-Queens Problem



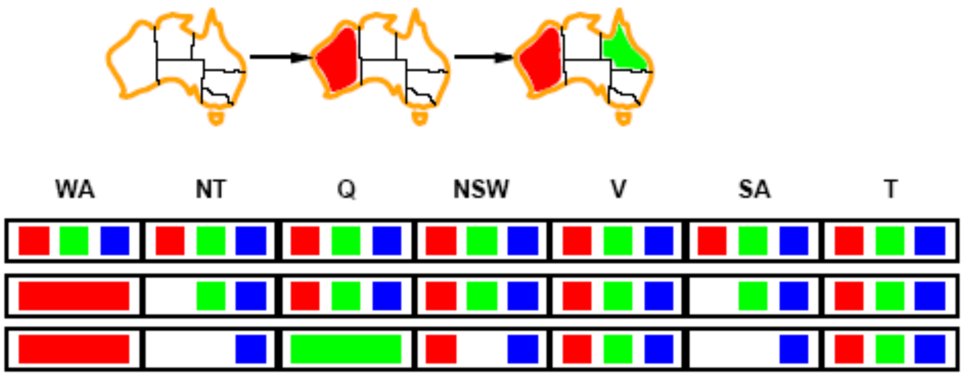


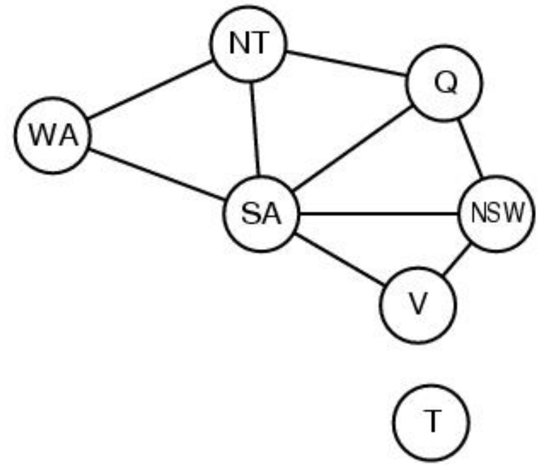
Example: 4-Queens Problem





Constraint propagation





- Solving CSPs with combination of heuristics plus forward checking is more efficient than either approach alone

- Forward checking checking does not detect all failures.
  - E.g., NT and SA cannot be blue

Constraint propagation

- Techniques like Constraint Propagation (CP) and Forward Checking (FC) are in effect eliminating parts of the search space
  - Somewhat complementary to search
- Constraint propagation goes further than FC by repeatedly enforcing constraints locally
  - Needs to be faster than actually searching to be effective
- Arc-consistency (AC) is a systematic procedure for constraining propagation (don’t worry about details)

Trade-offs

- Running stronger consistency checks…
  - Takes more time
  - But will reduce branching factor and detect more inconsistent partial assignments

  - No "free lunch"

Local search for CSPs

- Use complete-state representation
  - Initial state = all variables assigned values
  - Successor states = change 1 (or more) values
- For CSPs
  - allow states with unsatisfied constraints (unlike backtracking)
  - operators **reassign** variable values
  - hill-climbing with n-queens is an example
- Variable selection: randomly select any conflicted variable
- Value selection: *min-conflicts heuristic*
  - Select new value that results in a minimum number of conflicts with the other variables

Local search for CSP

function MIN-CONFLICTS(*csp, max_steps*) **return** solution or failure
  inputs: *csp*, a constraint satisfaction problem
      max_steps, the number of steps allowed before giving up

  current ← an initial complete assignment for *csp*
  for *i* = 1 to *max_steps* **do**
      if *current* is a solution for *csp* then return *current*
      var ← a randomly chosen, conflicted variable from VARIABLES[*csp*]
      value ← the value *v* for *var* that minimize CONFLICTS(*var,v,current,csp*)
      set *var* = *value* in *current*
  return *failure*

Advantages of local search

- Local search can be particularly useful in an online setting
  - Airline schedule example
    - E.g., mechanical problems require than 1 plane is taken out of service
    - Can locally search for another “close” solution in state-space
    - Much better (and faster) in practice than finding an entirely new schedule

- The runtime of min-conflicts is roughly independent of problem size.
  - Can solve the millions-queen problem in roughly 50 steps.

  - Why?
    - n-queens is easy for local search because of the relatively high density of solutions in state-space