Constraint Satisfaction Problems

Dr. Daisy Tang

Standard Search Problems

- Previous search problems:
  - Problems can be solved by searching in a space of states
  - state is a "black box" – any data structure that supports successor function, heuristic function, and goal test – problem-specific

- Constraint satisfaction problem
  - states and goal test conform to a standard, structured and simple representation
  - general-purpose heuristic

Constraint Satisfaction Problems (CSP)

- **<u>CSP</u> is defined by 3 components (X, D, C):**
  - **state**: a set of **variables** X, each X_i, with values from **domain** D_i
  - **goal test**: a set of **constraints** C, each C_i involves some subset of the variables and specifies the allowable combinations of values for that subset
  - Each constraint C_i consists of a pair **<scope, rel>**, where scope is a tuple of variables and rel is the relation, either represented explicitly or abstractly
- **X1 and X2 both have the domain {A, B}**
  - Constraints:
    - <(X1, X2), [(A, B), (B, A)]>, or
    - <(X1, X2), X1 ≠ X2>

Solution

- Each state in a CSP is defined by an assignment of values to some or all of the variables
- An assignment that does not violate any constraints is called a consistent or legal assignment
- A complete assignment is one in which every variable is assigned
- A solution to a CSP is consistent and complete assignment
- Allows useful general-purpose algorithms with more power than standard search algorithms

Example: Map Coloring

- Variables: X = {*WA, NT, Q, NSW, V, SA, T* }
- Domains: D_i = {red, green, blue}
- Constraints: adjacent regions must have different colors
- Solution?

Solution: Complete and Consistent Assignment

- Variables: X = {*WA, NT, Q, NSW, V, SA, T* }
- Domains: D_i = {red, green, blue}
- Constraints: adjacent regions must have different colors
- Solution? {WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = red}.

Constraint Graph

- Constraint graph: nodes are variables, arcs are constraints
- Binary CSP: each constraint relates two variables
- CSP conforms to a standard pattern
  - a set of variables with assigned values
  - generic successor function and goal test
 - generic heuristics
  - reduce complexity
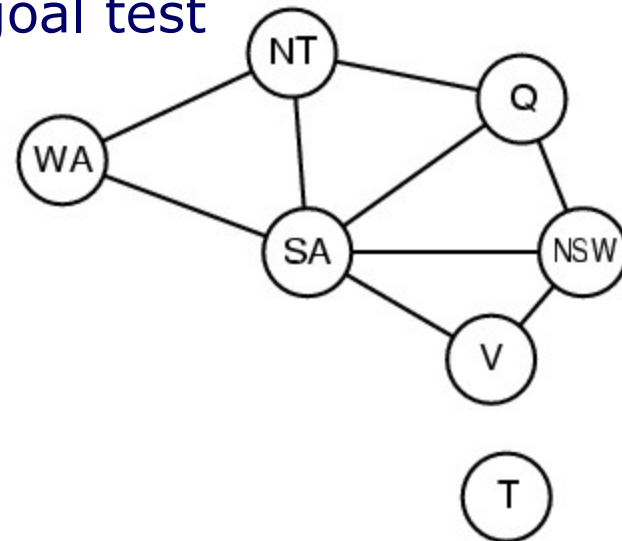
CSP as a Search Problem

- **Initial state**:
  - {} – all variables are unassigned
- **Successor function**:
  - a value is assigned to one of the unassigned variables with no conflict
- **Goal test**:
  - a complete assignment
- **Path cost**:
  - a constant cost for each step
- Solution appears at depth n if there are n variables
- Depth-first or local search methods work well

CSP Solvers Can be Faster

- CSP solver can quickly eliminate large part of search space

- If {SA = blue}

- Then 3^5 assignments can be reduced to 2^5 assignments, a reduction of 87%
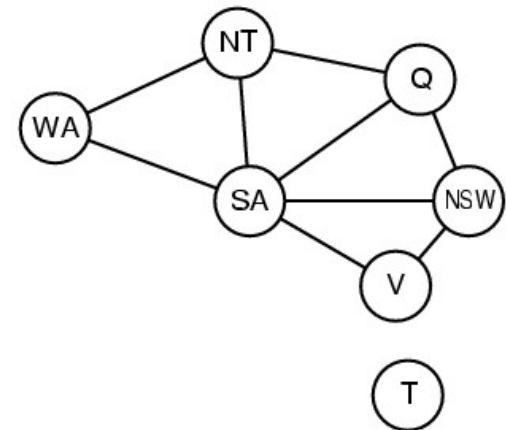
- In a CSP, if a partial assignment is not a solution, we can immediately discard further refinements of it

Exercise (1)

- Consider the problem of crossword puzzle: fitting words into a rectangular grid. Assume that a list of words is provided and that the task is to fill in the blank squares using any subset of the list. Formulate this problem precisely in two ways:

  - As a general search problem. Choose an appropriate search algorithm.

  - As a constraint satisfaction problem.

  - Word vs. letters
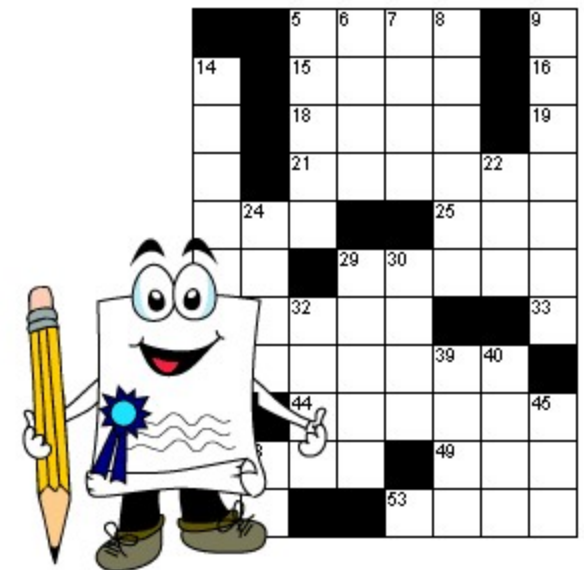
Exercise (2)

- Problem formulation as CSP:
  - Class scheduling: There is a fixed number of professors and classrooms, a list of classes to be offered, and a list of possible time slots for classes. Each professor has a set of classes that he or she can teach.

Types of Variables

- Discrete variables
  - finite domains:
    - n variables, domain size $d \rightarrow O(d^n)$ complete assignments
    - e.g., Boolean CSPs, such as 3-SAT (NP-complete)
    - Worst case, can't solve finite-domain CSPs in less than exponential time
  - 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

Types 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

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**

What Search Algorithm to Use?

- Since we can formulate CSP problems as standard search problems, we can apply search algorithms from chapter 3 & 4

- If breadth-first search were applied
  - branching factor? nd
  - tree size? $nd * (n\text{-}1)d * \ldots * d = n! * d^n$ leaves
  - complete assignments? d^n

- A crucial property to all CSPs: commutativity
  - the order of application of any given set of actions has no effect on the outcome
  - Variable assignments are commutative, i.e., [WA = red then NT = green] same as [NT = green then WA = red]

Backtracking Search

- Only need to consider assignments to a single variable at each node → $b = d$ and there are d^n leaves

- Backtracking search is used for a depth-first search that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign

- Backtracking search is the basic uninformed algorithm for CSPs

Backtracking Search Algorithm

```
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 ← 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 ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failue then return result
            remove { var = value } from assignment
    return failure
```
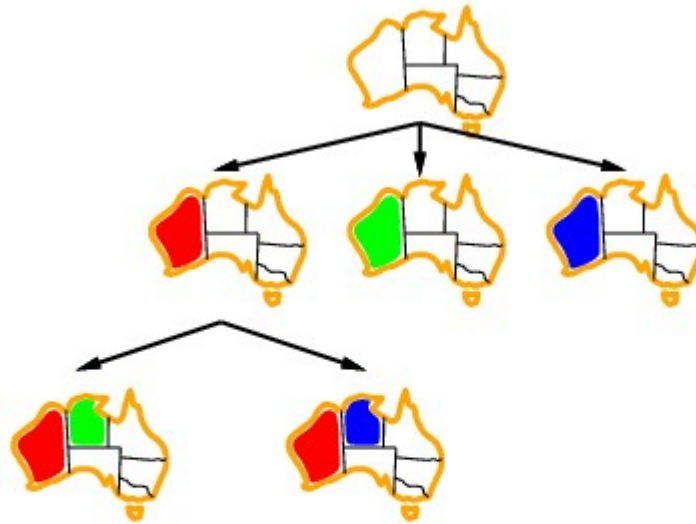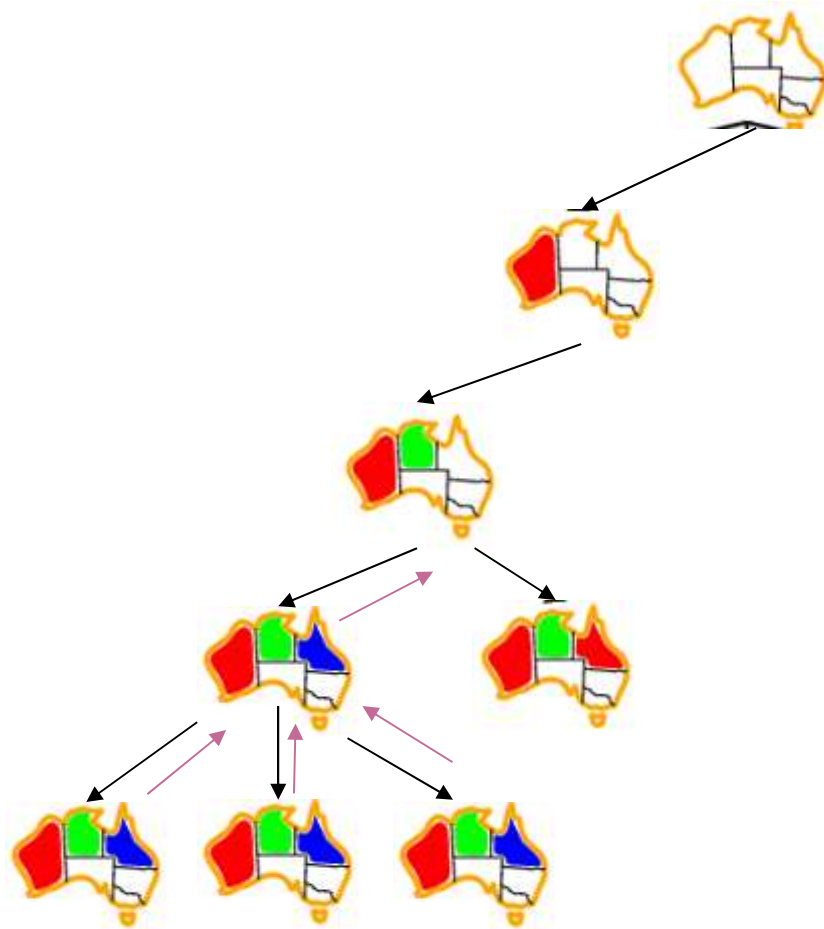
Backtracking Example

Backtracking Example
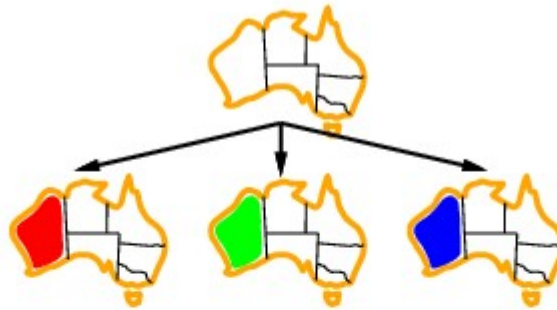
Backtracking Example

Backtracking Example

# Improving Backtracking Efficiency

- We can solve CSPs efficiently without domain-specific knowledge, addressing the following questions:

  - in what order should variables be assigned, values be tried?

  - what are the implications of the current variable assignments for the other unassigned variables?

  - when a path fails, can the search avoid repeating this failure?

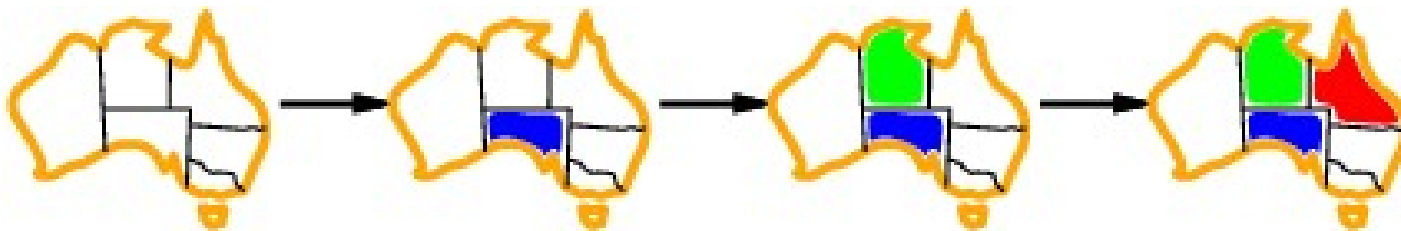# Variable and Value Ordering 1/3

- **Minimum remaining values** (MRV)
  - choose the variable with the fewest "legal" values
  - also called most constrained variable or fail-first heuristic
 - does it help in choosing the first variable?
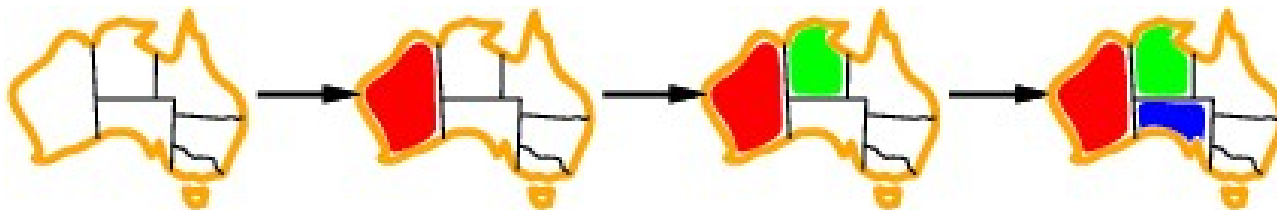
Variable and Value Ordering 2/3

- **Most constraining variable**:
  - selecting the variable that has the largest number of constraints on other unassigned variables
  - also called degree heuristics
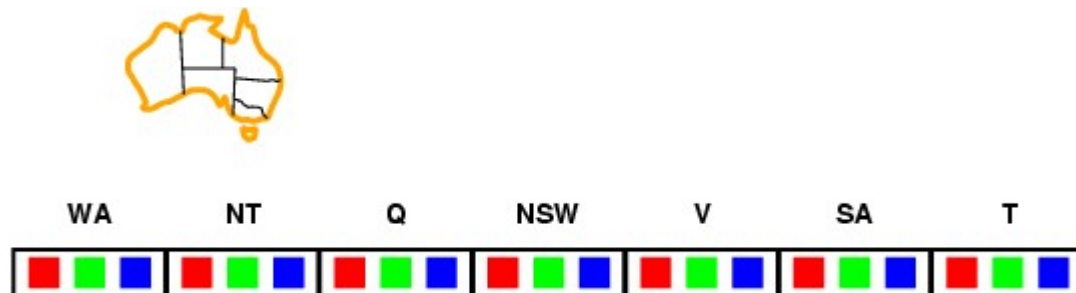- **Tie-breaker** among MRV

Variable and Value Ordering 3/3

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

Allows 1 value for SA

Allows 0 values for SA
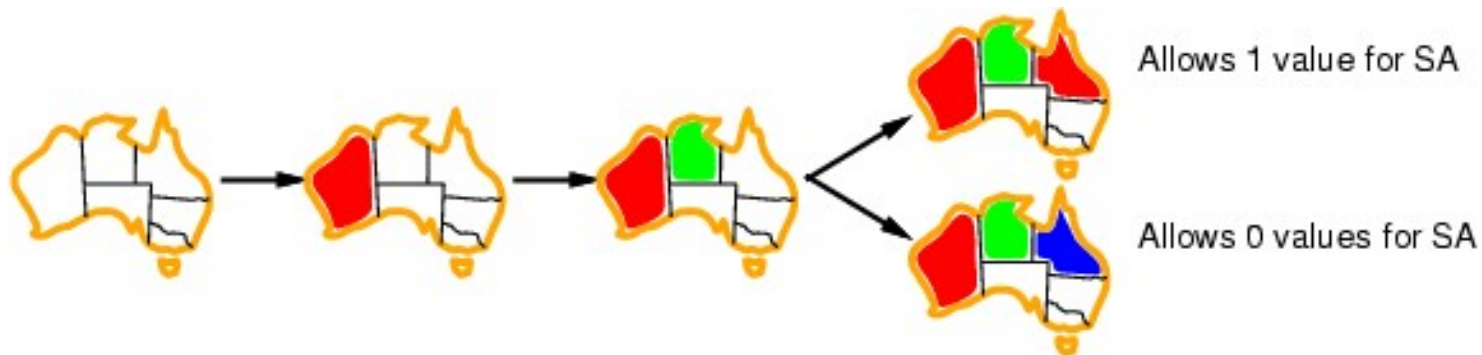
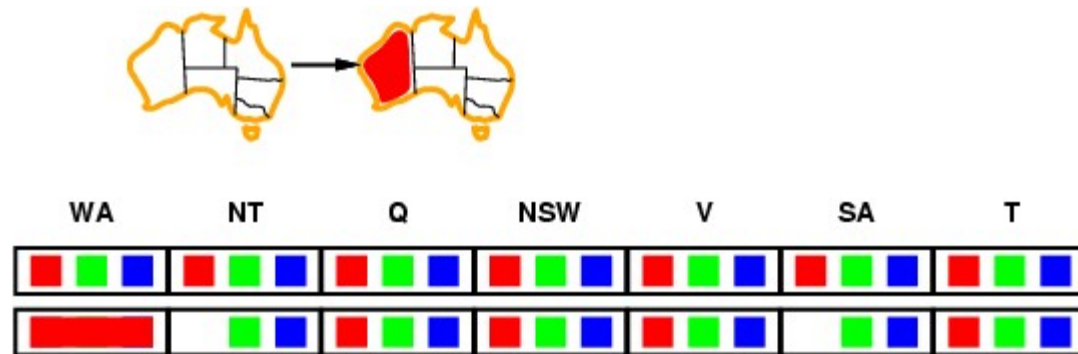- Combining these heuristics makes 1000 queens feasible

Propagating Information through Constraints

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

WA	NT	Q	NSW	V	SA	T

Propagating Information through Constraints

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

WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦

Propagating Information through Constraints

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

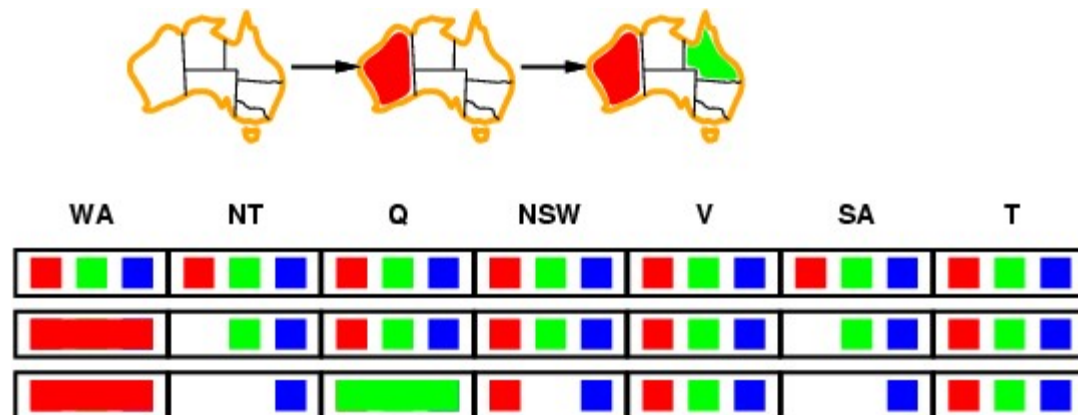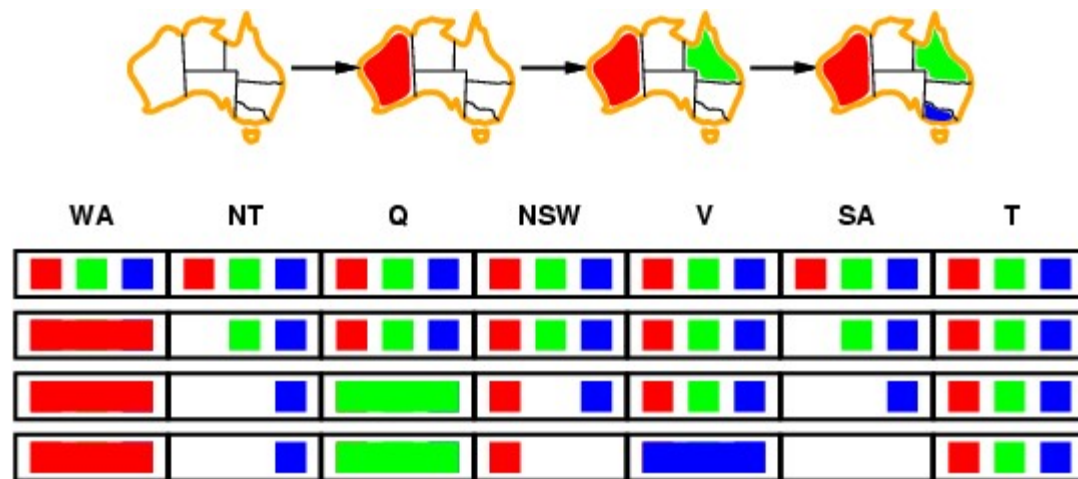# Propagating Information through Constraints

- ## Forward checking:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values

Propagating Information through Constraints

- Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:

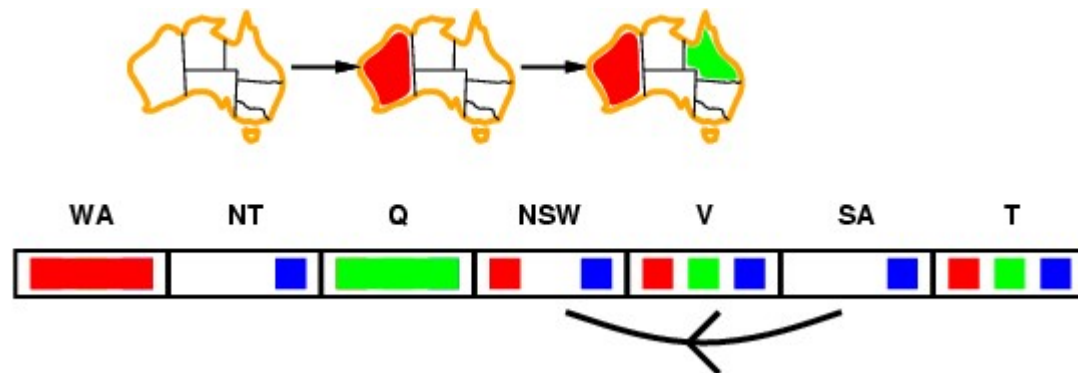| WA	NT	Q	NSW	V	SA	T
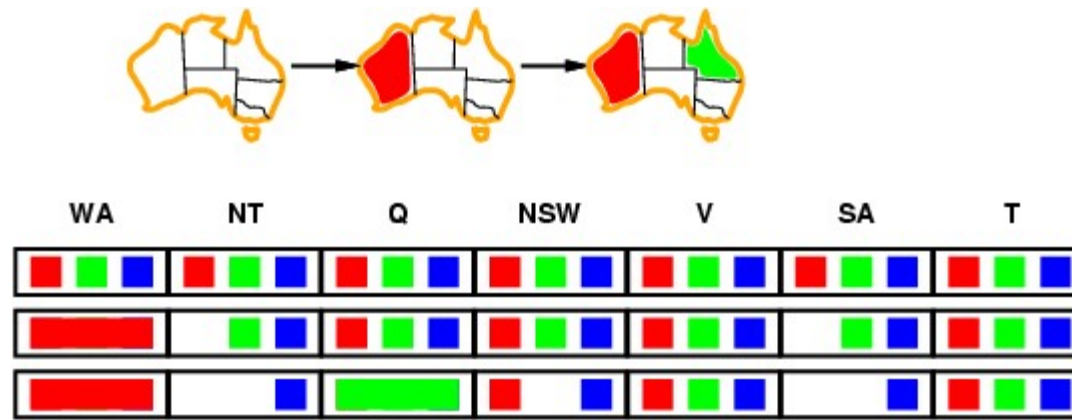
- NT and SA cannot both be blue!

- Constraint propagation repeatedly enforces constraints locally by propagating implications of a constraint of one variable onto other variables

Node Consistency

- A single variable is node-consistent if all the values in the variable's domain satisfy the variable's unary constraints

- For example, SA dislikes green

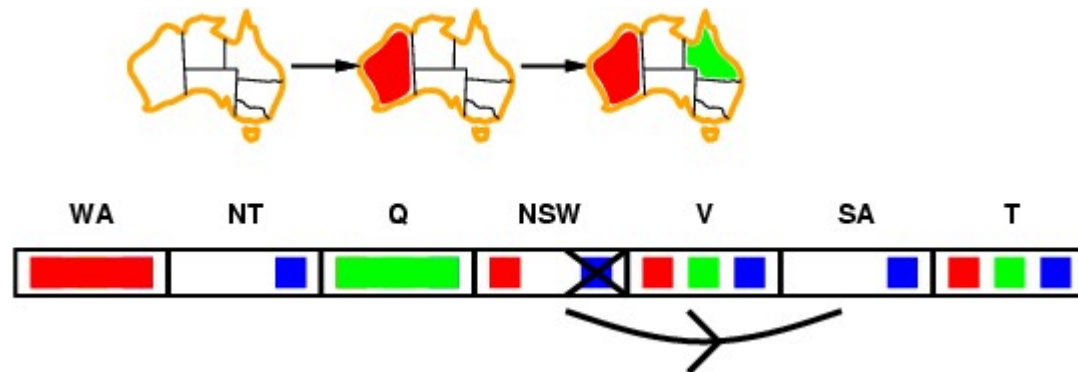- A network is node-consistent if every variable in the network is node-consistent

Arc Consistency

- Simplest form of propagation makes each arc consistent
- $X \rightarrow Y$ 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
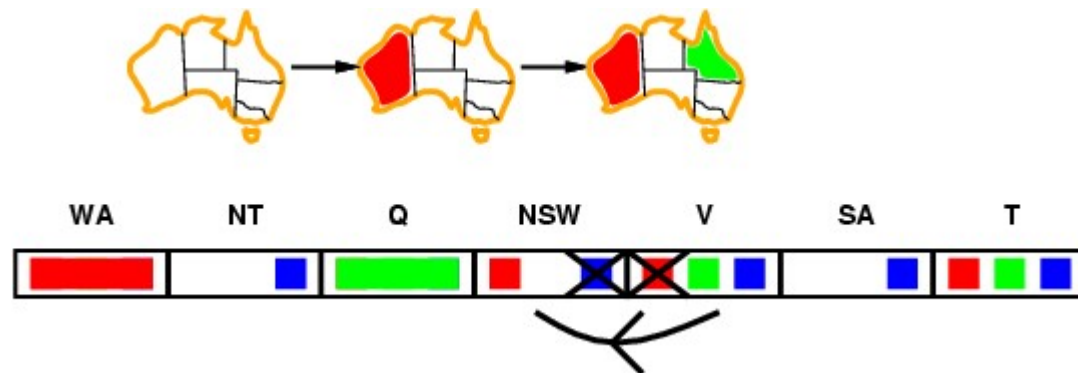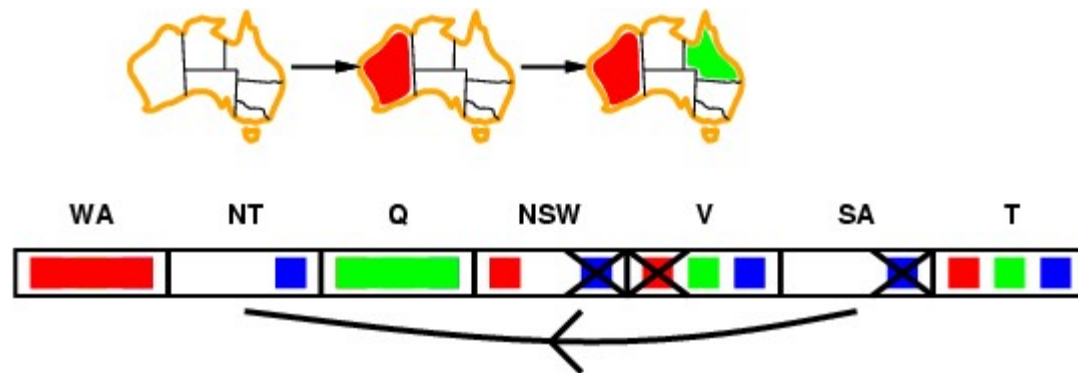   for every value *x* of *X* there is some allowed *y*

Arc Consistency

- Simplest form of propagation makes each arc consistent
- $X \rightarrow Y$ 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
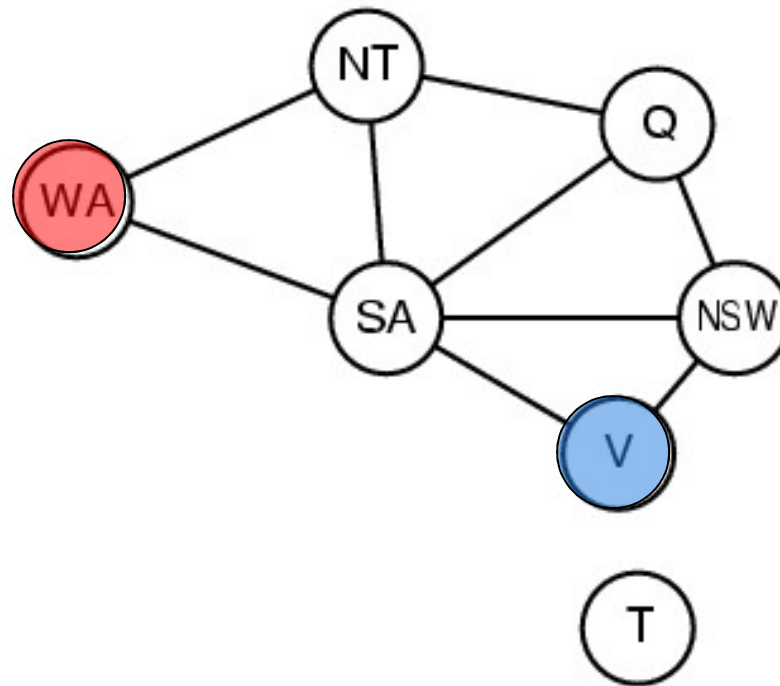
AC-3 Algorithm

- Time complexity O (n^2d^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 all the arcs in csp

    while $queue$ is not empty do
        $(X_i, X_j) \leftarrow$ REMOVE-FIRST($queue$)
        if RM-INCONSISTENT-VALUES(X_i, X_j) then
            for each X_k in NEIGHBORS[X_i] do
                add (X_k, X_i) to $queue$

function RM-INCONSISTENT-VALUES(X_i, X_j) returns true iff remove a value
    $removed \leftarrow false$
    for each x in DOMAIN[X_i] do
        if no value y in DOMAIN[X_j] allows (x,y) to satisfy constraint(X_i, X_j)
            then delete x from DOMAIN[X_i]; $removed \leftarrow true$
    return $removed$

# Example

- Use the AC-3 algorithm to show that arc consistency is able to detect the inconsistency of the partial assignment {WA = red, V = blue} in the following map coloring problem.

One Possible Trace

- One possible trace of the algorithm:
  - remove SA-WA, delete R from SA
  - remove SA-V, delete B from SA, leaving only G
  - remove NT-WA, delete R from NT
  - remove NT-SA, delete G from NT, leaving only B
  - remove NSW-SA, delete G from NSW
  - remove NSW-V, delete B from NSW, leaving only R
  - remove Q-NT, delete B from Q
  - remove Q-SA, delete G from Q
 - remove Q-NSW, delete R from Q, leaving no proper assignment for Q
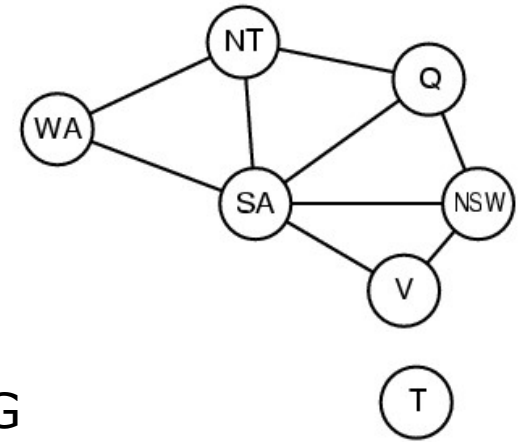
Path Consistency

- Consider map-coloring with only two colors
- Every arc is consistent initially
- Check the set {WA, SA} path consistenty with respect to NT

- More generally, k-consistency
  - 1-consistency = node consistency
  - 2-consistency = arc consistency
  - 3-consistency = path consistency

Local Search for CSPs

- Hill-climbing, simulated annealing 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 $h(n)$ = total number of violated constraints

Example: 4-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

h = 5   h = 2   h = 0
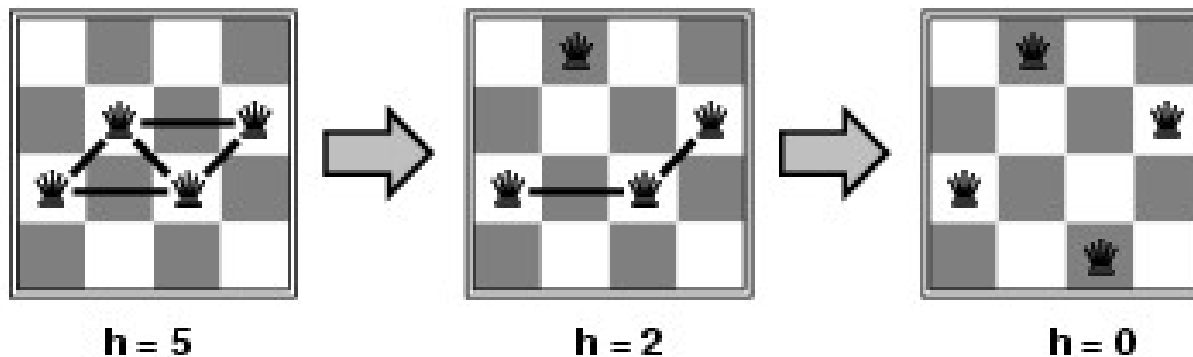
8-Queen Example
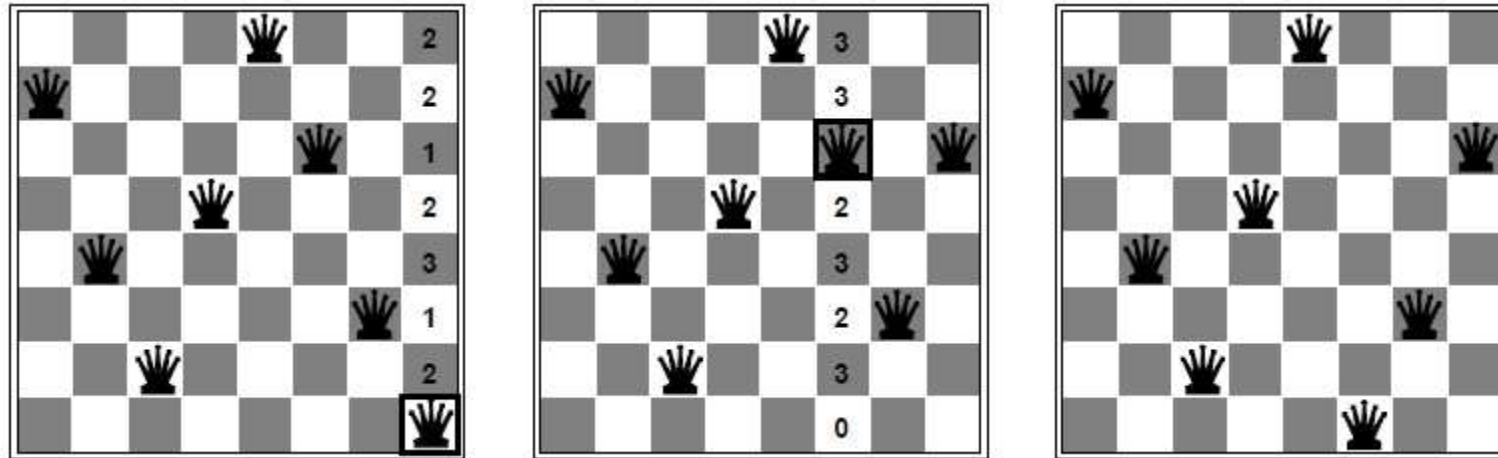
Min-conflicts is quite effective for many CSPs.

Given a random initial state, can solve n-queens in almost constant time for arbitrary n with high probability (e.g., $n = 10,000,000$)

Min-Conflicts Algorithms for CSP

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

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

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) does additional work to constrain values and detect inconsistencies

- Iterative min-conflicts is usually effective in practice