Introduction to Artificial Intelligence

Lecture: Constraint Satisfaction Problems

Outline

- Constraint Satisfaction Problems (CSPs)
- Constraint Propagation: Inference in CSPs
- Backtracking Search for CSPs
- Local Search for CSPs

Constraint Satisfaction Problems (CSPs)

- A constraint satisfaction problem (CSP) uses a factored representation for each state.
  - State = a set of variables and each of which has a value
  - Solution = each variable has a value that satisfies all constraints on that variable
- Take advantage of the structure of states
- General-purpose rather than problem-specific heuristics
  - Identify combinations of variable-value that violate the constraints → eliminate large portions of the search space all at once
  - Solutions to complex problems

Constraint Satisfaction Problems (CSPs)

- A CSP consists of the following three components
  - $\boldsymbol{X} = \{X_1,.., X_n\}$: a set of variables
  - $\boldsymbol{D} = \{D_1,.., D_n\}$: a set of domains, one for each variable.
  - $\boldsymbol{C}$: a set of constraints that state allowable combinations of values.
- Each C consists of a pair <$scope$, rel>
  - $scope$: a tuple of variables that participate in the constraint
  - A relation rel defines the values that participated variables can take

Constraints in CSPs

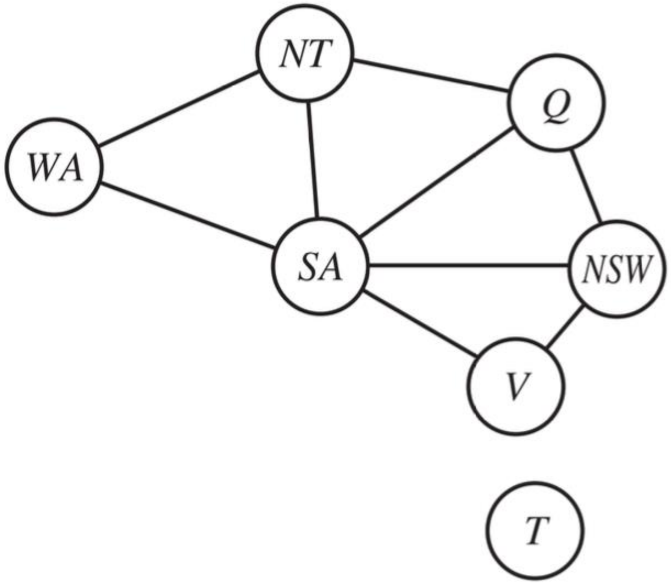
- Assume that both X_1 and X_2 have the domain $\{A, B\}$
- “Two variables must have different values”
- A relation can be an explicit list of all tuples of values that satisfy the constraint.
  - $<(X_1, X_2), \{(A, B), (B, A)\}>$
- It can be an abstract relation that supports two operations
  - Test whether a tuple is a member of the relation
  - Enumerate the members of the relation
  - $<(X_1, X_2), X_1 \neq X_2>$

Solutions for CSPs

- Each state is defined by an assignment of values to some or all the variables.
- A solution to a CSP is a **consistent** – **complete** assignment.
  - A **consistent** assignment does not violate any constraints.
  - A **complete** assignment has every variable assigned, while a **partial** assignment assigns values to only some variables.

Example: Map Coloring

- Each state is assigned a color in {red, green, blue}.
- Adjacent states have different colors.
- Constraint graph:
  - Nodes ⇔ Variables
  - Arcs ⇔ Constraints



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

$$\{SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW,$$

$$SA \neq V, WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V\}$$

  - $SA \neq WA$ is a shortcut of <(SA,WA), $SA \neq WA$>
  - $SA \neq WA$ can be fully enumerated as

{(red,green), (red,blue), (green,red),

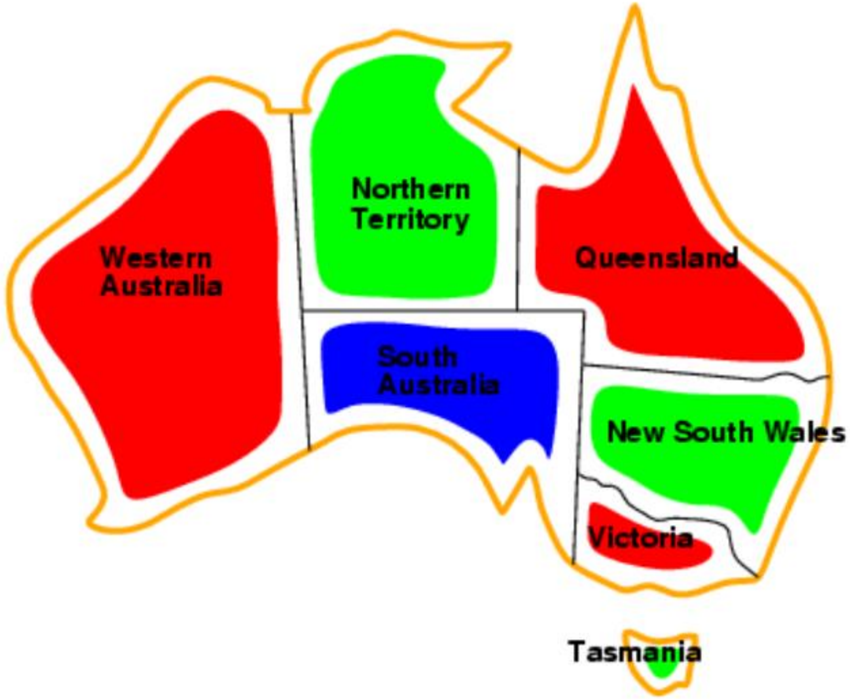
(green,blue), (blue,red), (blue,green)}

Example: Map Coloring

- There are many possible solutions to this problem.

$$\{WA = red, NT = green, Q = red, NSW = green,$$

$$V = red, SA = blue, T = red\}$$



Benefits of CSPs

- Provide natural representation for a wide variety of problems
- Many problems intractable in regular state-space search can be solved quickly with CSP formulation.
- Better insights to the problem and its solution.

Variations on the CSP formalism

- Discrete and finite variables
  - n variables, domain size $d \rightarrow O(d^n)$ complete assignments
  - E.g., map coloring, scheduling with time limits, 8-queens etc.
- Discrete, infinite domains
  - Sets of Integers, strings, etc.
  - E.g., job scheduling without deadlines
  - Constraint language: understand constraints without enumeration
- Continuous domains
  - Real-world problems often involve continuous domains and even real-valued variables.

CPSs in practise

- Operations Research (scheduling, timetabling)
  - Scheduling the time of observations on the Hubble Space Telescope
  - Airline schedules
- Linear programming
  - Constraints must be linear equalities or inequalities

    → solved in time polynomial in the number of variables.
- Bioinformatics (DNA sequencing)
- Electrical engineering (circuit layout-ing)
- Cryptography
- Computer vision: image interpretation

Types of constraints

- Unary constraint: restrict the value of a single variable
  - $SA \neq green$
- Binary constraint: relate two variables
  - $SA \neq WA$
- Higher-order constraints: involve three or more variables
  - E.g., Professors A, B, and C cannot be on a committee together
  - Always possible to be represented by multiple binary constraints
- Global constraints: involving an arbitrary number of variables
  - $Alldiff$ = all variables involved must have different values

Preference constraints

- Which solutions are preferred → soft constraints
  - E.g., red is better than $green$

    → this often can be represented by a cost for each variable assignment

- Constraint optimization problem (COP): a combination of optimization with CSPs → linear programming

Constraint propagation

- Constraints help to reduce the number of legal values for a variable

  → legal values for another variable are also reduced.

- Intertwined with search, or done as a preprocessing step.
- Enforcing local consistency in each part of a graph causes inconsistent values to be eliminated throughout the graph.

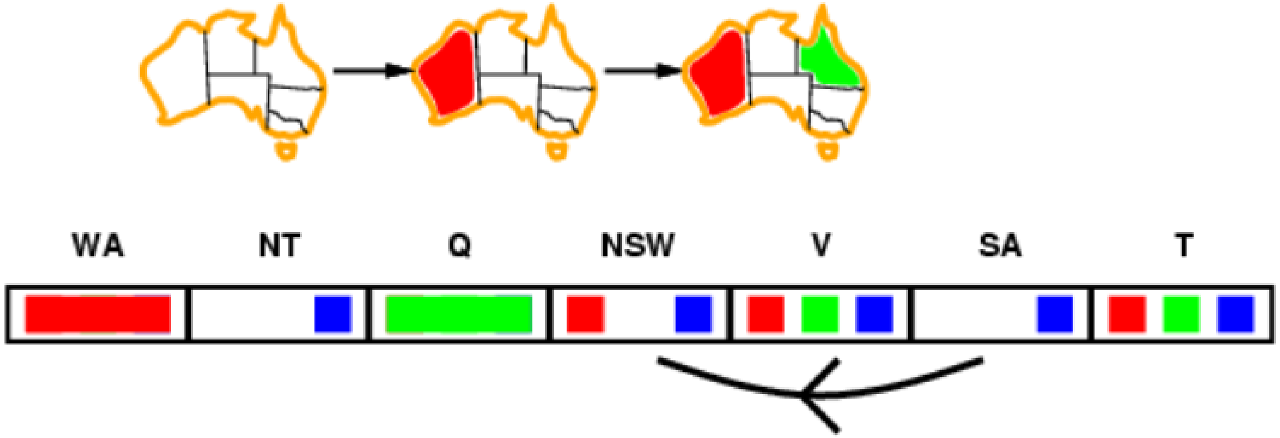
Node consistency

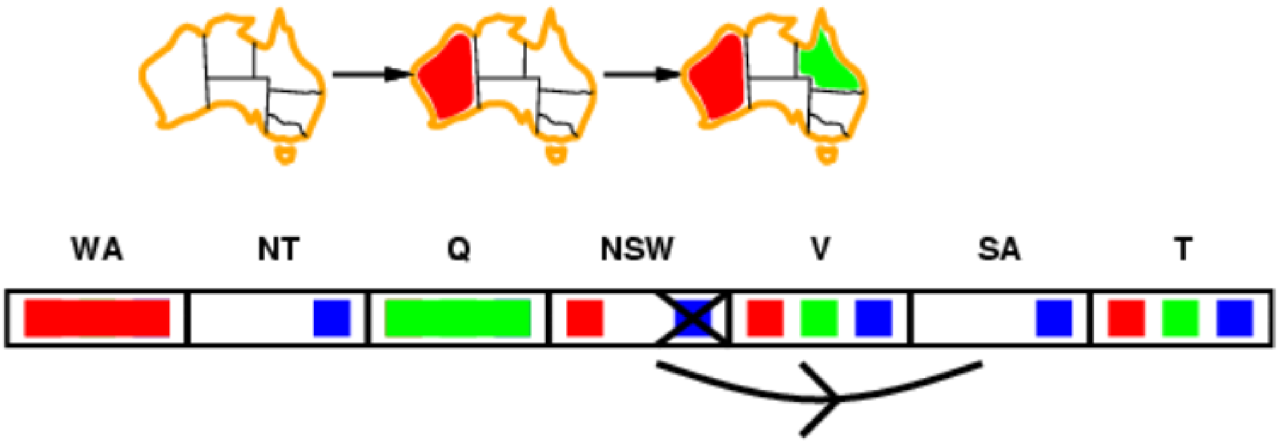
- A single variable is node-consistent if all the values in the variable's domain satisfy the variable's unary constraints.
- Eliminate all the unary constraints in a CSP by running node consistency.
- E.g., The South Australians dislike green, the domain of $\{SA\}$ will be $\{red, \cancel{green}, blue\}$

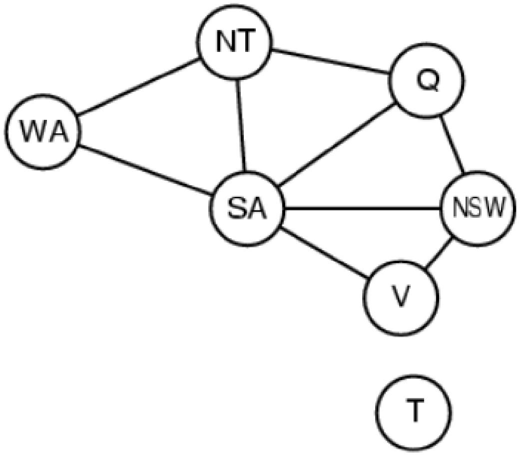
Arc consistency

- A variable in a CSP is arc-consistent if every value in its domain satisfies the variable's binary constraints.
- Arc consistency may have no effect in several cases.
- E.g., the Australia map, no matter what value chosen for SA (or for WA), there is a valid value for the other variable.

Arc consistency







Arc consistency

- Run as a preprocessor before the search starts or after each assignment
- AC must be run repeatedly until no inconsistency remains
- Need a systematic method for arc-checking
    - If X loses a value, neighbors of X need to be rechecked

      $\rightarrow$ incoming arcs can become inconsistent again while outgoing arcs stay still

AC-3 algorithm

function AC-3(csp)

returns false if an inconsistency is found and true otherwise

inputs: csp, a binary CSP with components (X, D, C)

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 REVISE(csp, Xi , Xj) **then**

if size of $D_i = 0$ **then return** false

for each X_k in X_i.NEIGHBORS - $\{X_j\}$ **do**

add (X_k , X_i) to queue

return true

AC-3 algorithm

function REVISE(csp, X_i , X_j)

returns true iff we revise the domain of Xi

revised ← false

for each x in D_i **do**

if no value y in D_j allows (x ,y) to satisfy the constraint between X_i and X_j **then**

delete x from D_i

revised ← true

return revised

- The worst-case complexity is $O(cd^3)$
  - n: number of variables, each has domain size d, c binary constraints (arc)

Backtracking Search

- States are defined by the values assigned so far
  - Initial state: empty assignment { }
  - Successor function: assign a value to an unassigned variable that agrees with the current assignment

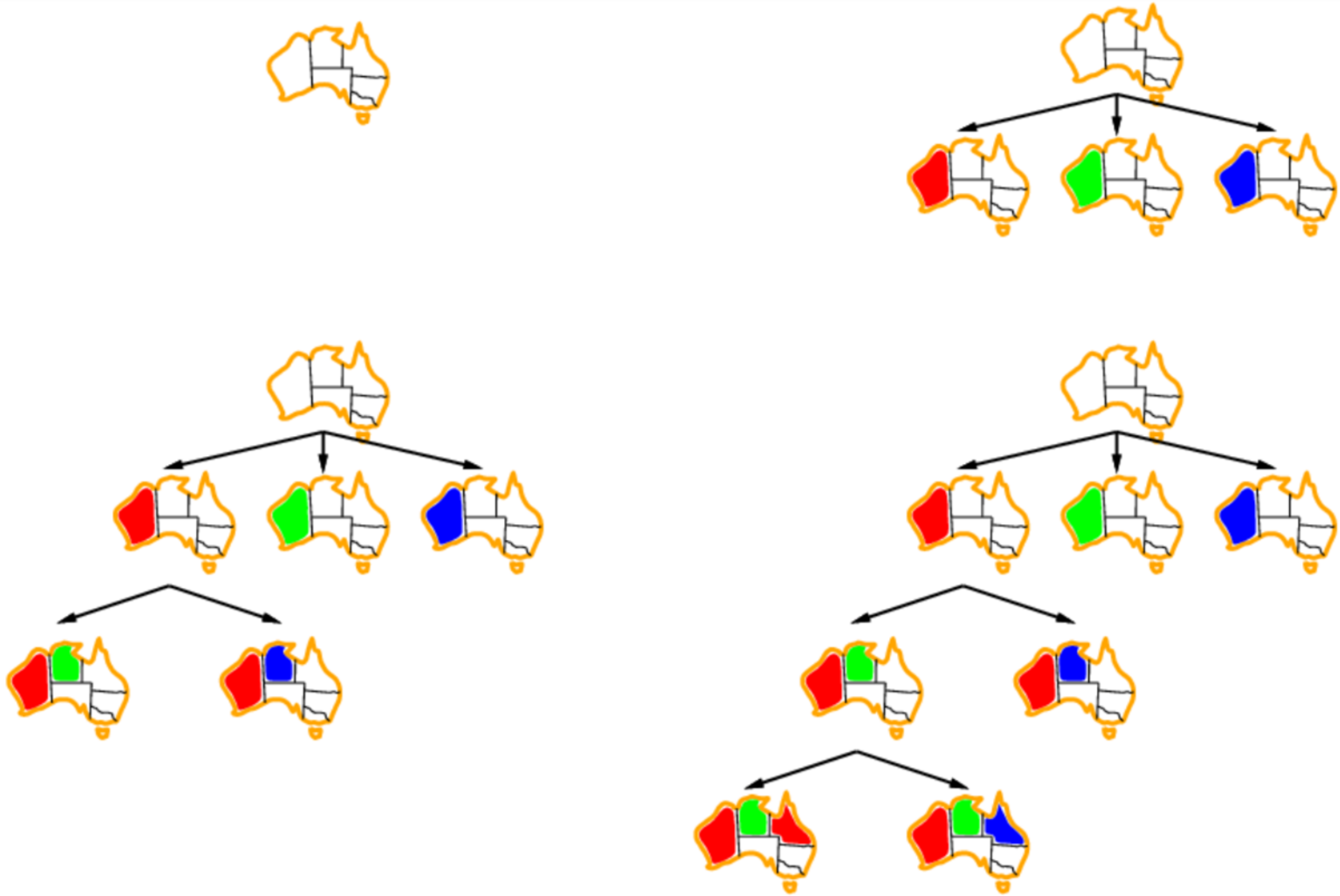
    → fail if no legal assignments

  - Goal test: the current assignment is complete
- Depth-first search: choose values for one variable at a time and backtrack when a variable has no legal values left

Backtracking Search

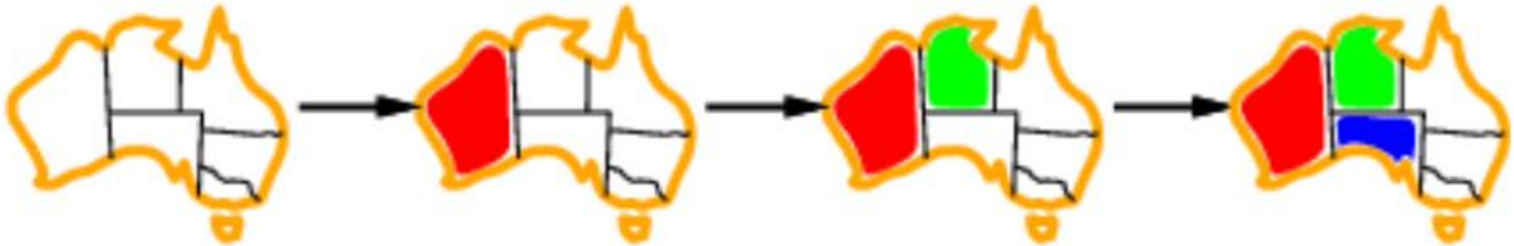
```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
    return BACKTRACK({ }, csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment then
            add {var = value} to assignment
            inferences ← INFERENCE(csp, var, value)
            if inferences !=  failure then
                add inferences to assignment
                result ← BACKTRACK(assignment, csp)
                if result != failure then
                    return result
        remove {var = value} and inferences from assignment
    return failure
```

Backtracking Search



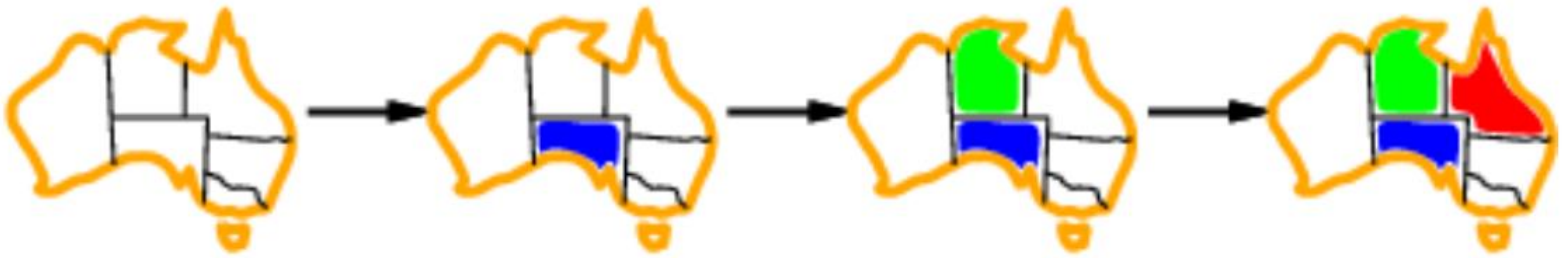
Variable and value ordering

- Minimum-remaining-values (MRV) heuristic: choose the variable with the fewest legal values
  - E.g., after [$WA = red$, $NT = green$] only one possible value for SA



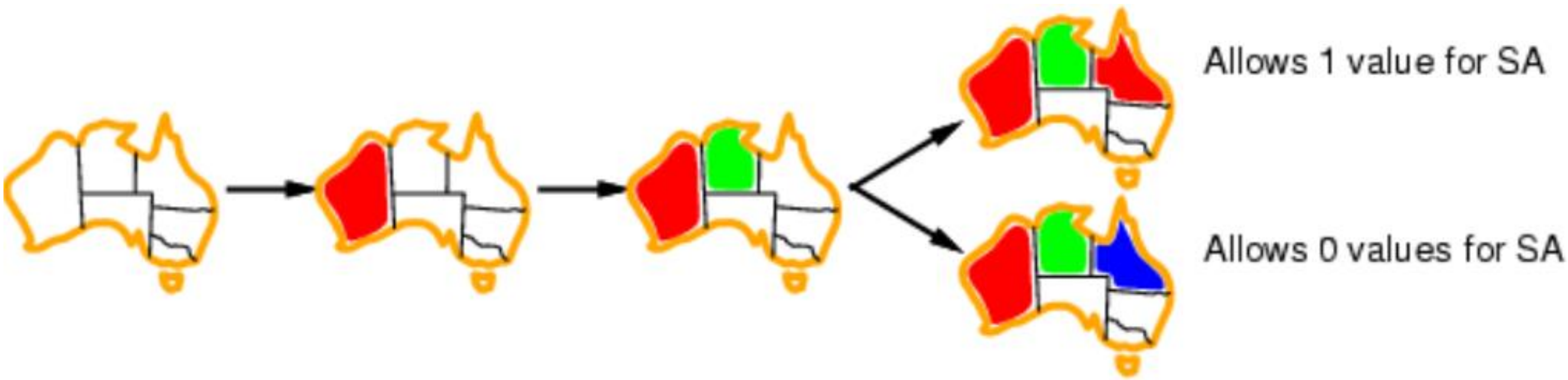
Variable and value ordering

- Degree heuristic (DH): choose the variable that involves in the largest number of constraints on other unassigned variables.
  - E.g., SA has a highest degree of 5



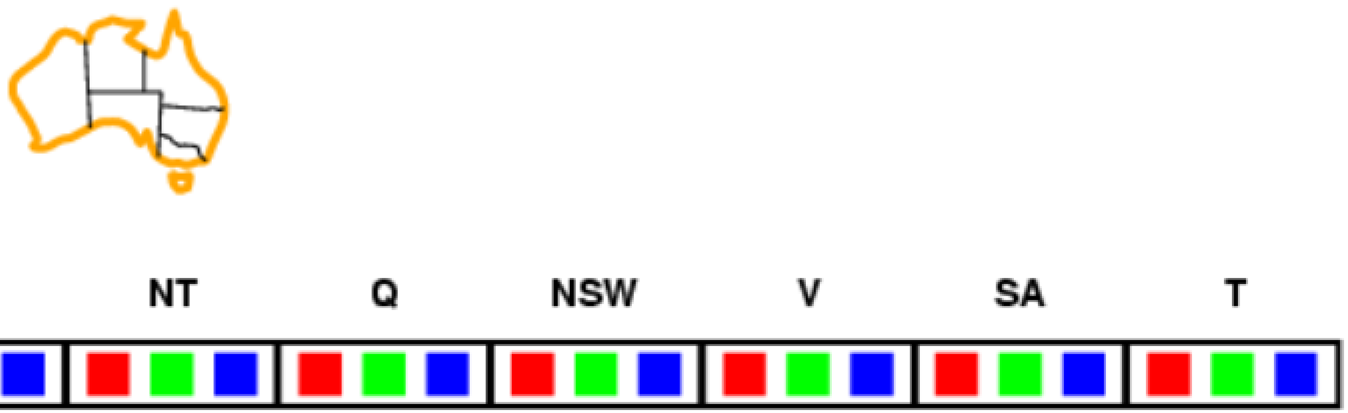
Variable and value ordering

- Least constraining value (LCV) heuristic: given a variable, choose the value that leaves the maximum flexibility for subsequent variable assignments



Inference: Forward checking

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



Local search for CSPs

- Complete-state formulation
  - The initial state assigns a value to every variable → violation
  - The search changes the value of one variable at a time → eliminate the violated constraints
- Min-conflicts heuristic: the minimum number of conflicts with other variables
- Min-conflicts is surprisingly effective for many CSPs.

MIN-CONFLICTS algorithm

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 ← 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 csp.VARIABLES

    value ← the value v for var that minimizes CONFLICTS(var, v, current, csp)

    set var = value in current

  return failure

Local search for CSPs

- The landscape of a CSP under the min-conflicts heuristic usually has a series of plateaux.
- Plateau search: allow sideways moves to another state with the same score.
- Tabu search: keep a small list of recently visited states and forbid the algorithm to return to those states
- Simulated annealing can also be used.

Constraint weighting

- Concentrate the search on the important constraints
- Each constraint is given a numeric weight, W , initially all 1.
- At each step, choose a variable/value pair to change that has the lowest total weight of all violated constraints
- Increase the weight of each constraint that is violated by the current assignment.

Homework

- Conduct homework in the given notebook

References

- Stuart Russell and Peter Norvig. 2009. Artificial Intelligence: A Modern Approach (3rd ed.). Prentice Hall Press, Upper Saddle River, NJ, USA.
- Lê Hoài Bắc, Tô Hoài Việt. 2014. Giáo trình Cơ sở Trí tuệ nhân tạo. Khoa Công nghệ Thông tin. Trường ĐH Khoa học Tự nhiên, ĐHQG-HCM.
- Nguyễn Ngọc Thảo, Nguyễn Hải Minh. 2020. Bài giảng Cơ sở Trí tuệ Nhân tạo. Khoa Công nghệ Thông tin. Trường ĐH Khoa học Tự nhiên, ĐHQG-HCM.