Constraint Satisfaction Problems

These slides are primarily based on the slides created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley-http://ai.berkeley.edu.

The artwork is by Ketrina Yim.

Today

- Homework 2 / Homework 3
- Improving Hill Climbing
- CSP

Homework 2

Things to know and fix for homework 3:

- visited, closed or explored set: use a set not a list or a tuple. Membership testing is much more efficient for sets than lists or tuples.
- Follow the graph search algorithm from page 40 - lecture 6: a node should be expanded only if the corresponding state has not been visited before.

Graph Search Pseudo-Code

```
function GRAPH-SEARCH(problem, fringe) return a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE[node]) then return node
        if STATE[node] is not in closed then
            add STATE[node] to closed
            for child-node in EXPAND(STATE[node], problem) do
                fringe ← INSERT(child-node, fringe)
            end
    end
```

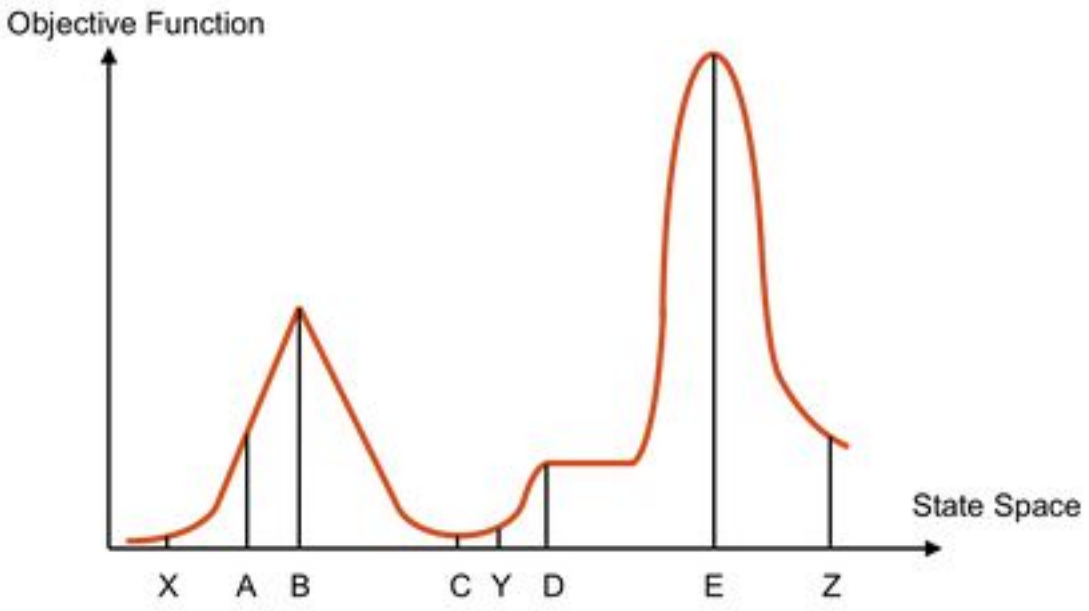
Improving Hill Climbing

```
function HILL-CLIMBING( problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node

    current ← MAKE-NODE(INITIAL-STATE[problem])
    loop do
        neighbor ← a highest-valued successor of current
        if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
        current ← neighbor
    end
```

Hill Climbing

- Starting from X, we end up in B
- Starting from Y, we end up in D
- Starting from Z, we end up in E



Hill Climbing

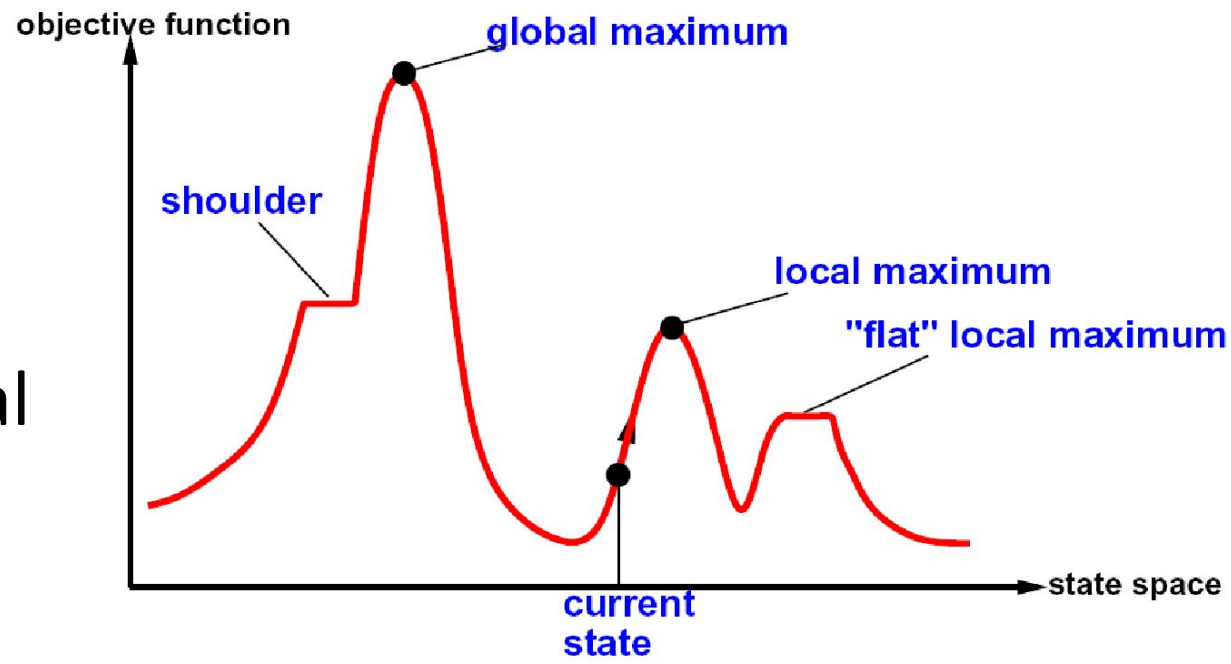
What's bad about this approach?

- Complete?
- Optimal?

What's good about it?

For some problems, a non optimal solution is ok

Let's take a look at some possible improvements



Random-restart Hill Climbing

Idea: Overcome local maxima by conducting several hill-climbing searches, starting from a random initial state.

We keep the best result.

This is a surprisingly effective algorithm in many cases.

Simulated Annealing

- Idea: Escape local maxima by allowing random downhill moves
  - But make them rarer as time goes on

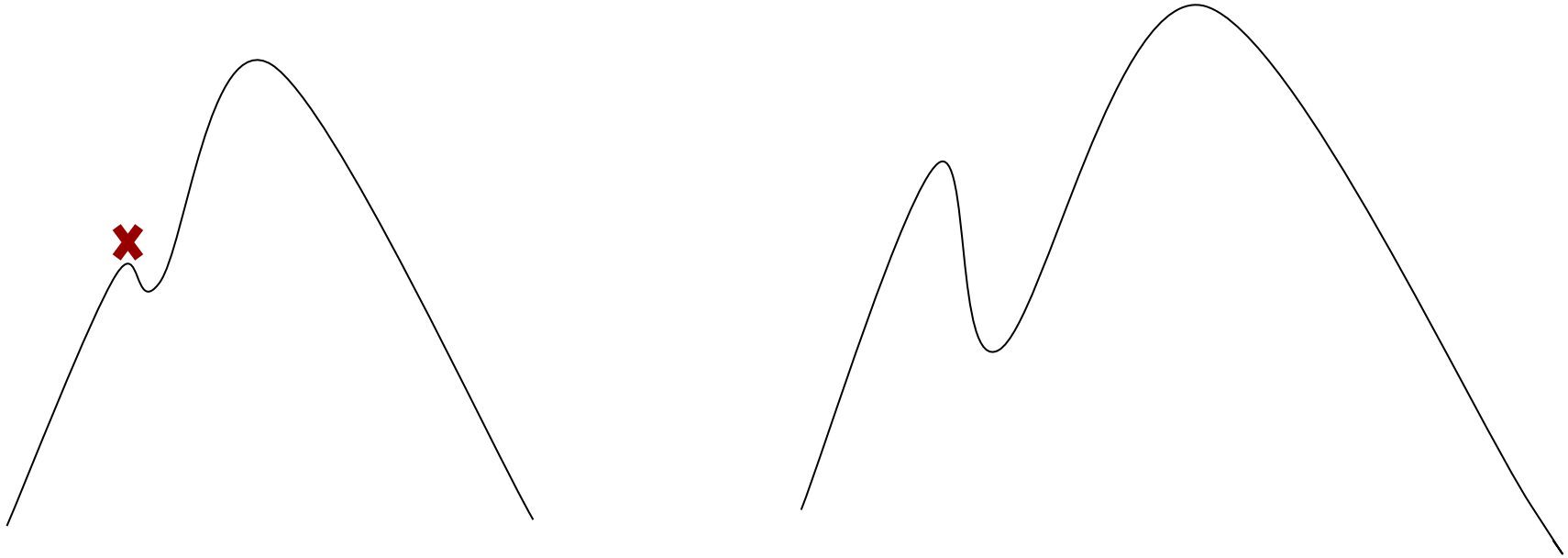
```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
          schedule, a mapping from time to "temperature"
  local variables: current, a node
                   next, a node
                   T, a "temperature" controlling prob. of downward steps

  current ← MAKE-NODE(INITIAL-STATE[problem])
  for t ← 1 to ∞ do
      T ← schedule[t]
      if T = 0 then return current
      next ← a randomly selected successor of current
      ΔE ← VALUE[next] – VALUE[current]
      if ΔE > 0 then current ← next
      else current ← next only with probability e^(Δ E/T)
```

If T is decreased slowly enough, it will converge to optimal state.

Simulated Annealing

- Idea: Escape local maxima by allowing random downhill moves
  - But make them rarer as time goes on



The more downhill steps we need to escape a local optimum, the less likely we are to ever make them all in a row.

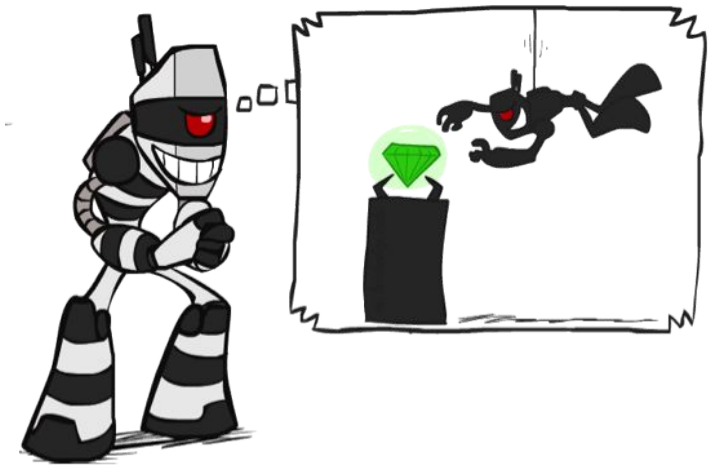
Constraint Satisfaction Problems

Assumptions about the world: a single agent, deterministic, fully observable, discrete

What is Search For?

Standard Search Problems: sequence of actions

- The path to the goal is the important thing
- Paths have various costs, depths
- Heuristics give problem-specific guidance



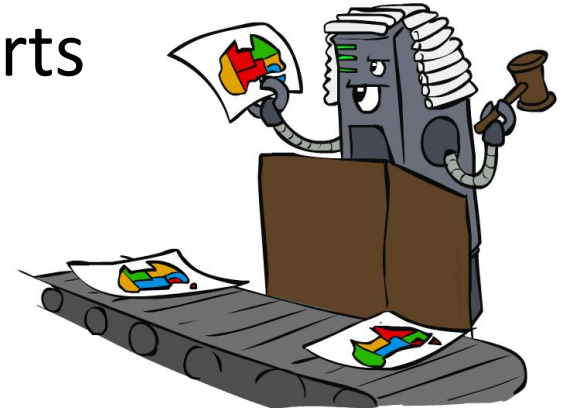
Identification Problems: assignments to variables

- The goal itself is important, not the path
- All paths at the same depth (for some formulations)
- CSPs are specialized for identification problems

Constraint Satisfaction Problems

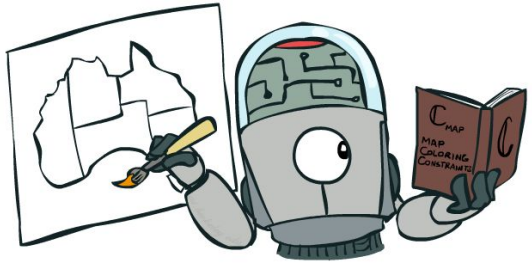
Standard search problems:

- State is a "black box", any data structure that supports goal test and successor function
- State is atomic, indivisible



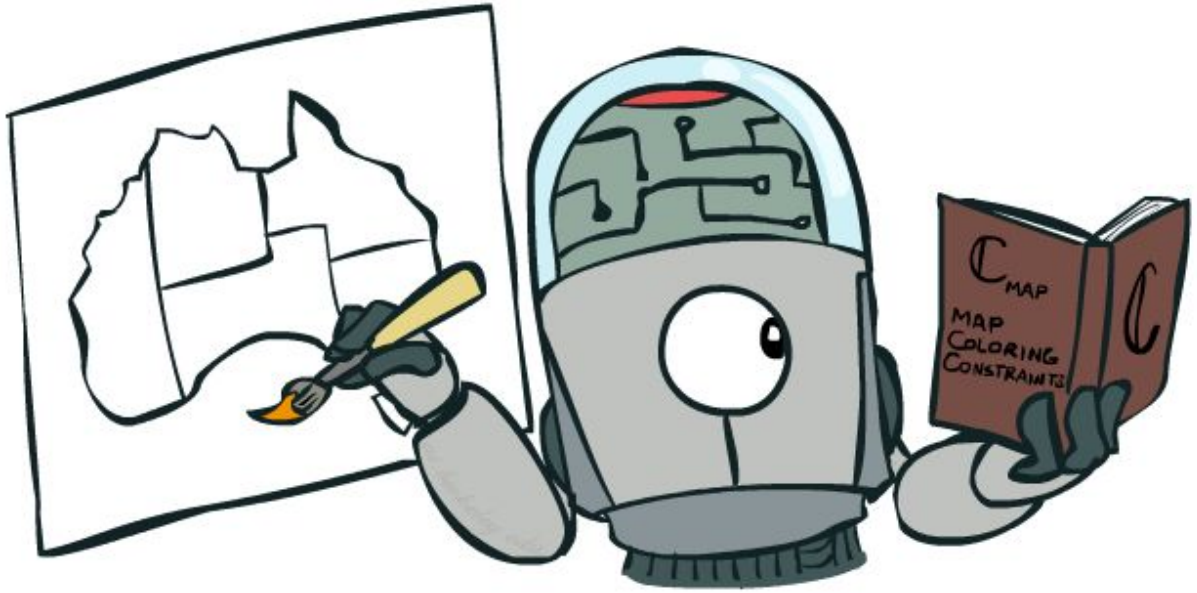
Constraint satisfaction problems (CSPs):

- Factored representation
- State is defined by a set of variables X_i with values from a domain D_i
- Goal test is a set of constraints C specifying allowable combinations of values for subsets of variables



Constraint Satisfaction Problems

Allow useful general-purpose algorithms/heuristics with more power than standard search algorithms.



Constraint Satisfaction Problems

A constraint satisfaction problem (CSP) consists of :

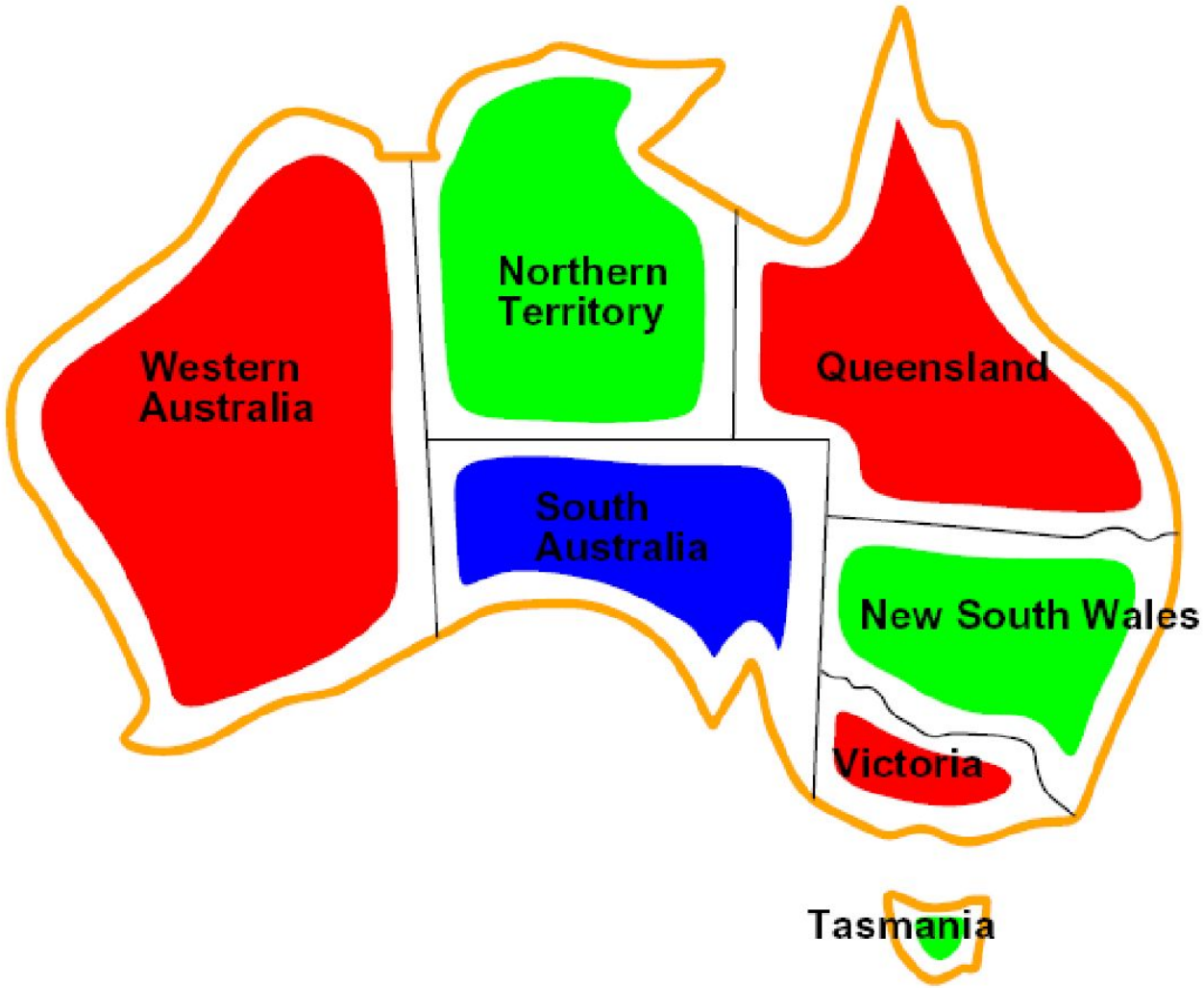
- X: a set of variables $\{X_1, \ldots, X_n\}$
- D: a set of domains $\{D_1, \ldots, D_n\}$, one for each variable

  Each domain D_i is a set of allowable values for variable X_i
- C: a set of constraints that specify allowable combinations of values

Constraint Satisfaction Problems

- Each state in a CSP is defined by an assignment of values to some or all of the variables.
- A consistent (or legal) assignment is an assignment that does not violate any constraints
- A complete assignment is one where each variable is assigned
- A solution is a consistent complete assignment

CSP Example


Western Australia
Northern Territory
Queensland
South Australia
New South Wales
Victoria
Tasmania

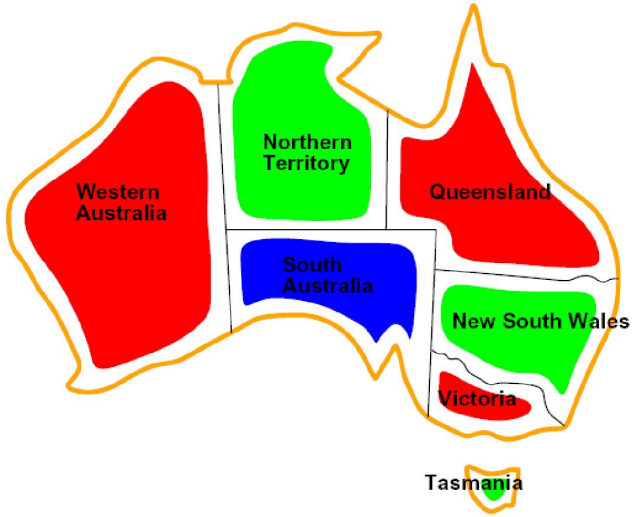
Example: Map Coloring

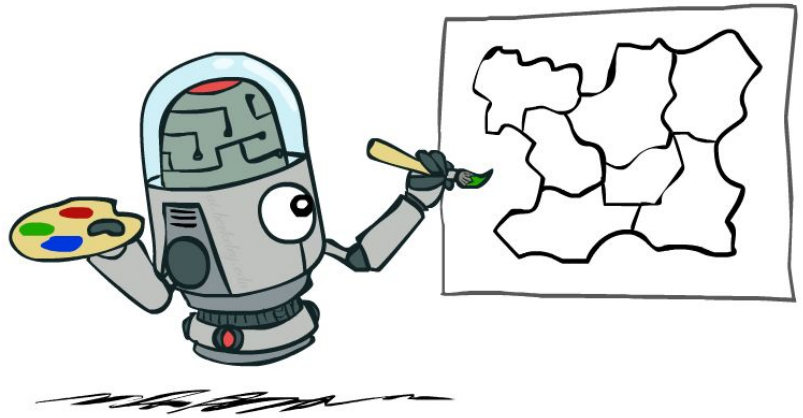
- Variables:

  X = {WA, NT, Q, NSW, V, SA, T}

- Domains:

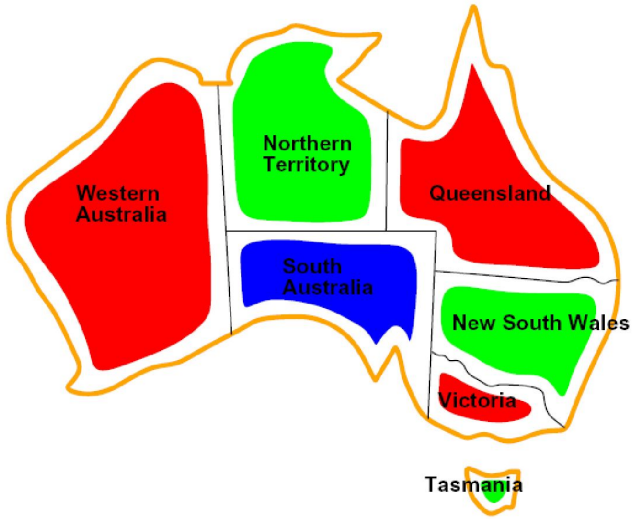
  Di = {red, green, blue}





Example: Map Coloring

Constraints: adjacent regions must have different colors - how many constraints?



Implicit constraints - provide code to compute

WA ≠ NT, WA ≠ SA, NT ≠ SA, NT ≠ Q, SA ≠ Q,

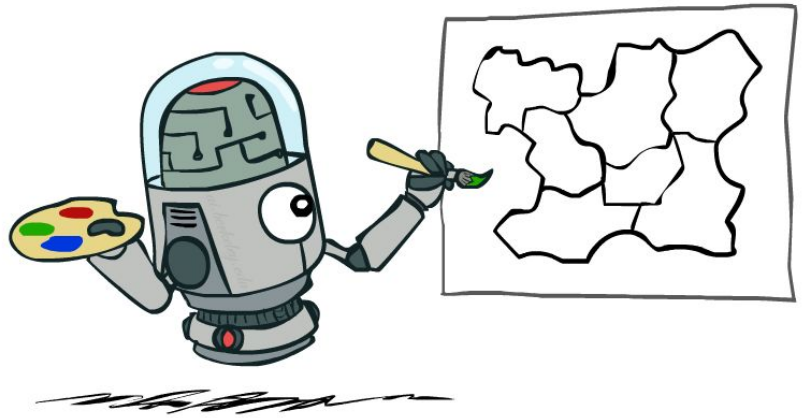
Q ≠ NSW, V ≠ NSW, V ≠ SA, NSW ≠ SA

Explicit constraints - list of legal tuples:

(WA, NT) ∈ {(red, green),(red, blue),(green, red),(green, blue), (blue, red), (blue, green)}

(WA, SA) ∈ {(red, green),(red, blue),...

...

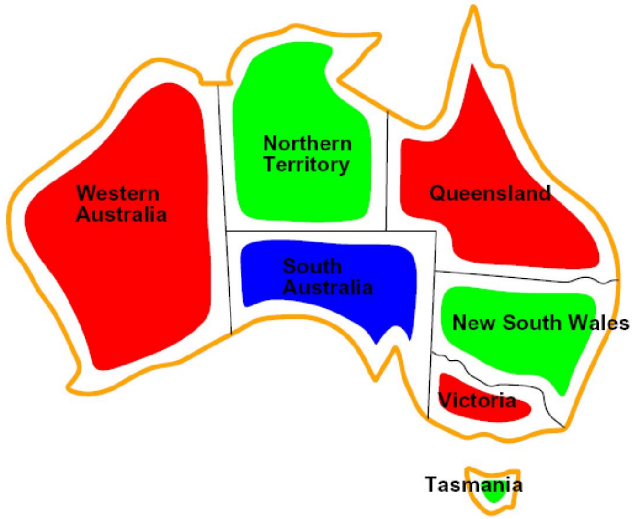


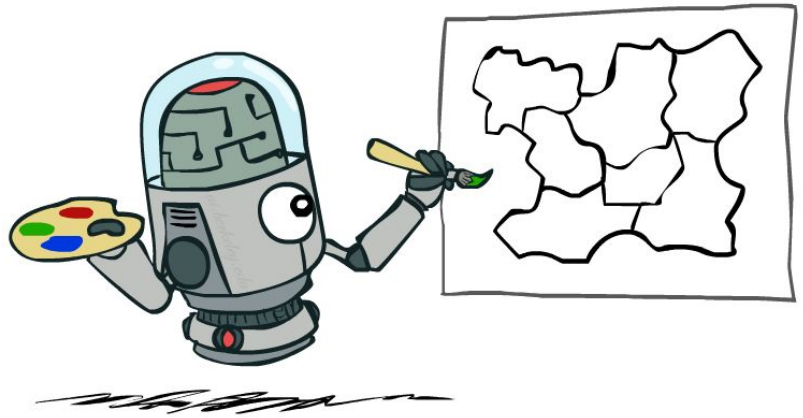
Example: Map Coloring

Solutions are assignments satisfying all constraints. Example:

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

V = red, SA = blue, T = green}





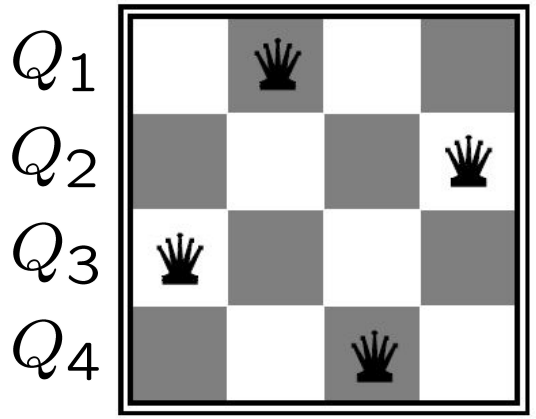
Example: 4-Queens

Variables:

Q_1, Q_2, Q_3, Q_4

Domains:

{*1, 2, 3, 4*}



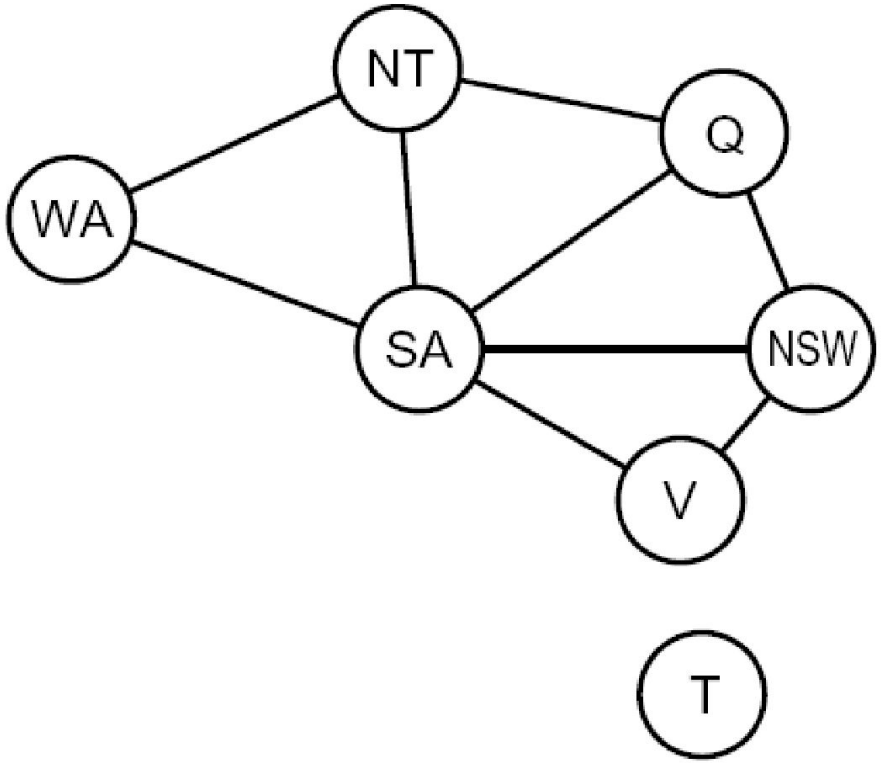
Constraints:

$(Q_1, Q_2) \in \{(1, 3), (1, 4), (2,4), (3,1), (4, 1), (4, 2)\}$

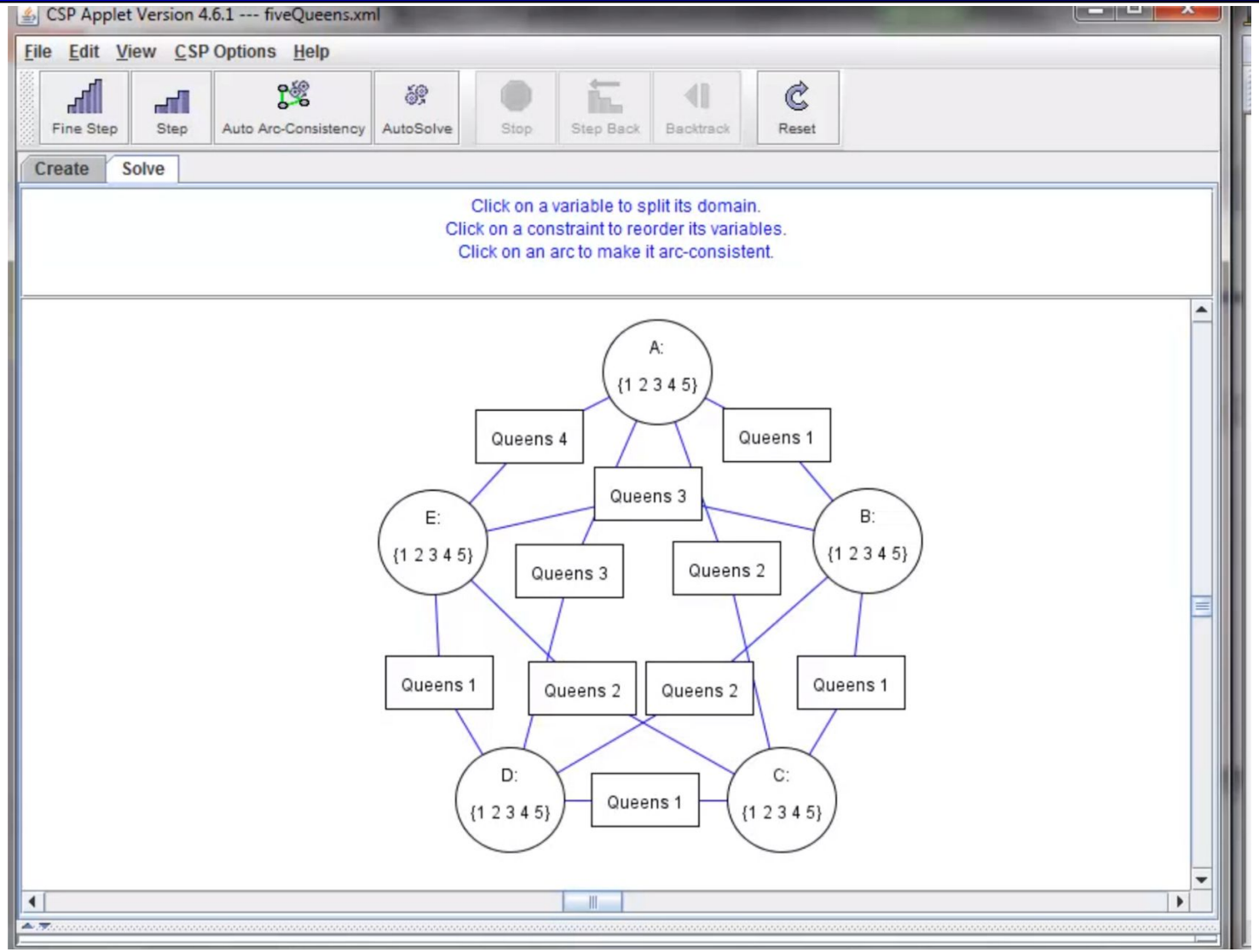
...

Constraint Graphs

- Binary CSP: each constraint relates at most two variables
- Binary constraint graph: nodes are variables, arcs show constraints
- General-purpose CSP algorithms use the graph structure to speed up search. E.g., Tasmania is an independent subproblem!



Demo 5-Queens



	`1	2	3	4	5
A					
B					
C					
D					
E					

Queens 1, Queens 2, Queens 3 and Queens 4 are the constraints.

[Demo: CSP applet (made available by aispace.org) -- n-queens]

Example: Cryptarithmetic

- Variables:

  $F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$

- Domains:

  $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

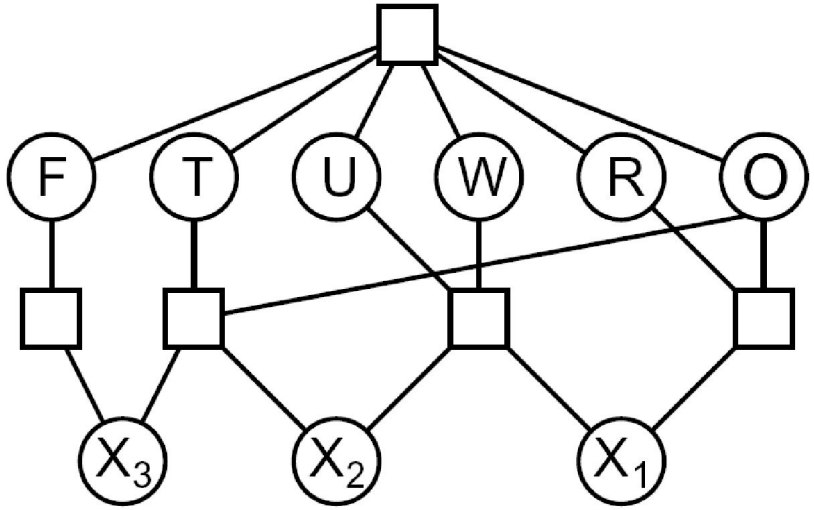
- Constraints:

  alldiff(F, T, U, W, R, O)

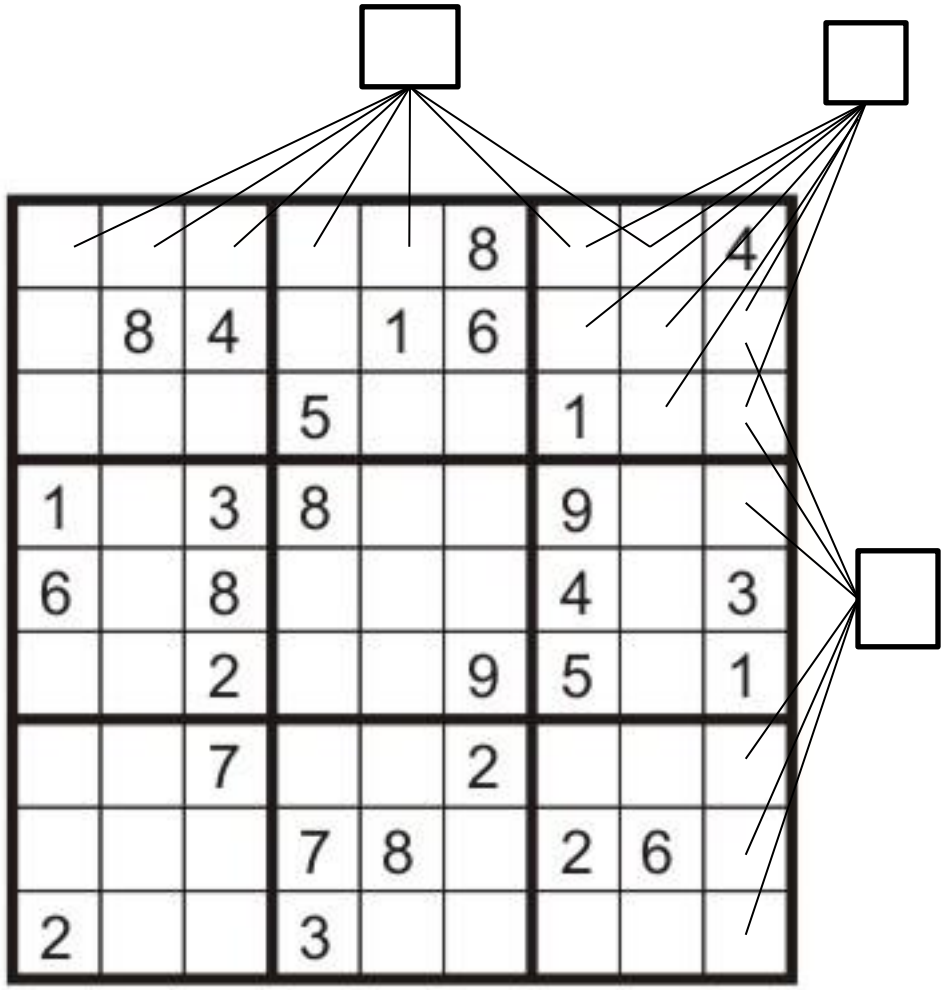
  $O + O = R + 10 \cdot X_1$

  $\ldots$

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



Example: Sudoku



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

  9-way alldiff for each column

  9-way alldiff for each row

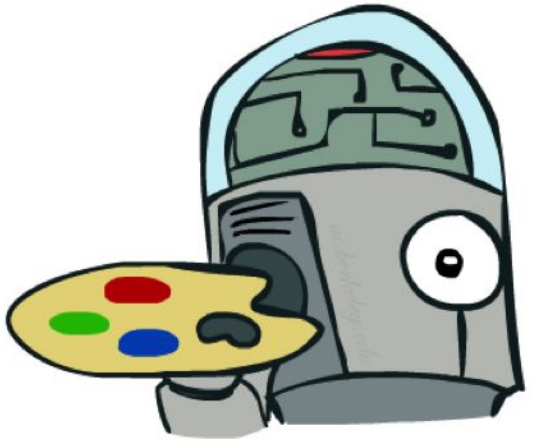
  9-way alldiff for each region

  (or can have a bunch of pairwise inequality constraints)

Varieties of CSPs

Discrete Variables

- Finite domains
  - Size d means $O(d^n)$ complete assignments (n variables)
  - Boolean CSPs, including Boolean satisfiability (NP-complete)



- Infinite domains (integers, strings, etc.)
  - E.g., job scheduling, variables are start/end times for each job - can no longer use explicit constraints

Varieties of CSPs

Continuous variables

- E.g., start/end times for Hubble Telescope observations
- Linear programming methods

Varieties of Constraints

Unary constraints involve a single variable: equivalent to reducing the corresponding domains: SA ≠ green

Binary constraints involve pairs of variables: SA ≠ WA

Higher-order constraints involve 3 or more variables: cryptarithmetic column constraints

Global constraints involve an arbitrary number of variables: Alldiff

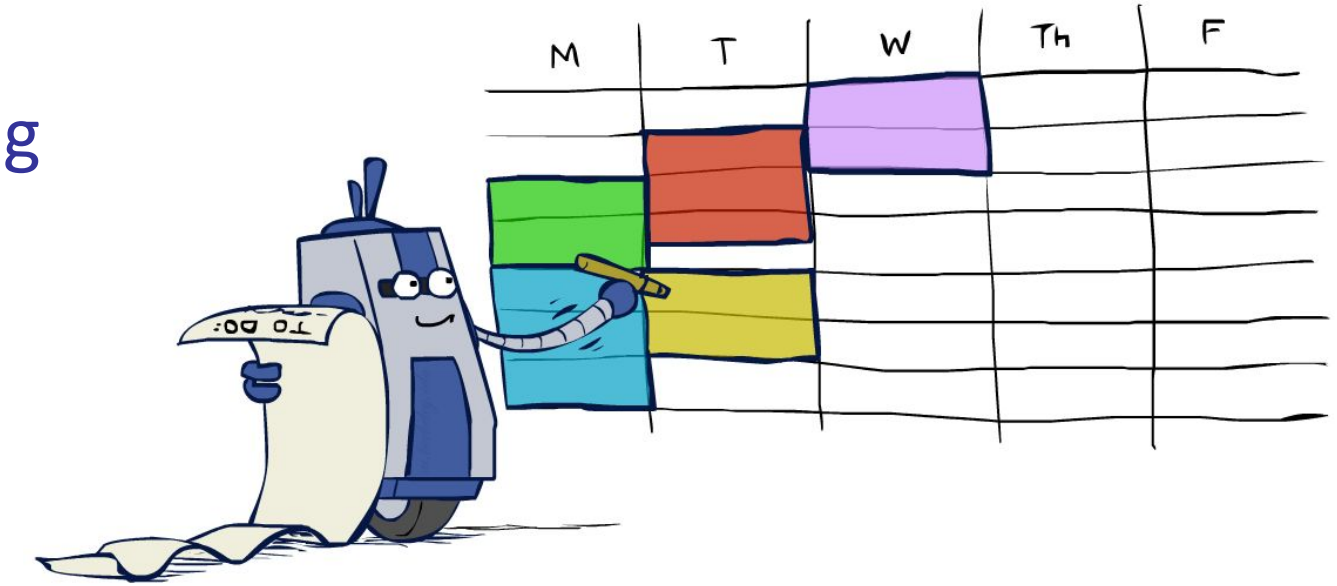
Varieties of Constraints

Preferences (soft constraints):

- E.g., red is better than green
- Often representable by a cost for each variable assignment
- Gives constrained optimization problems - not CSP

Real-World CSPs

- Assignment problems: e.g., who teaches what class
- Timetabling problems: e.g., which class is offered when and where?
- Hardware configuration
- Transportation scheduling
- Factory scheduling
- Circuit layout
- Fault diagnosis
- … lots more!



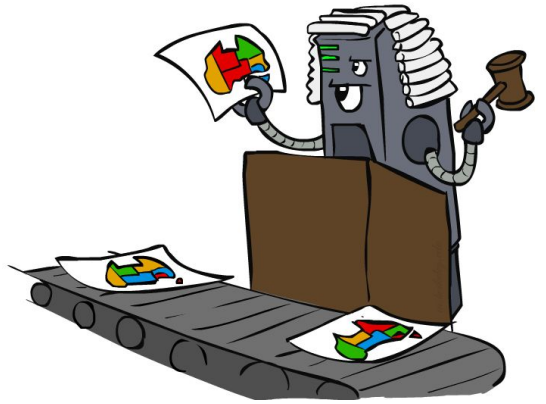
Solving CSPs

Standard Search Formulation of CSPs

We'll start with the straightforward, naïve approach, then improve it.

States are defined by the values assigned so far (partial assignments).

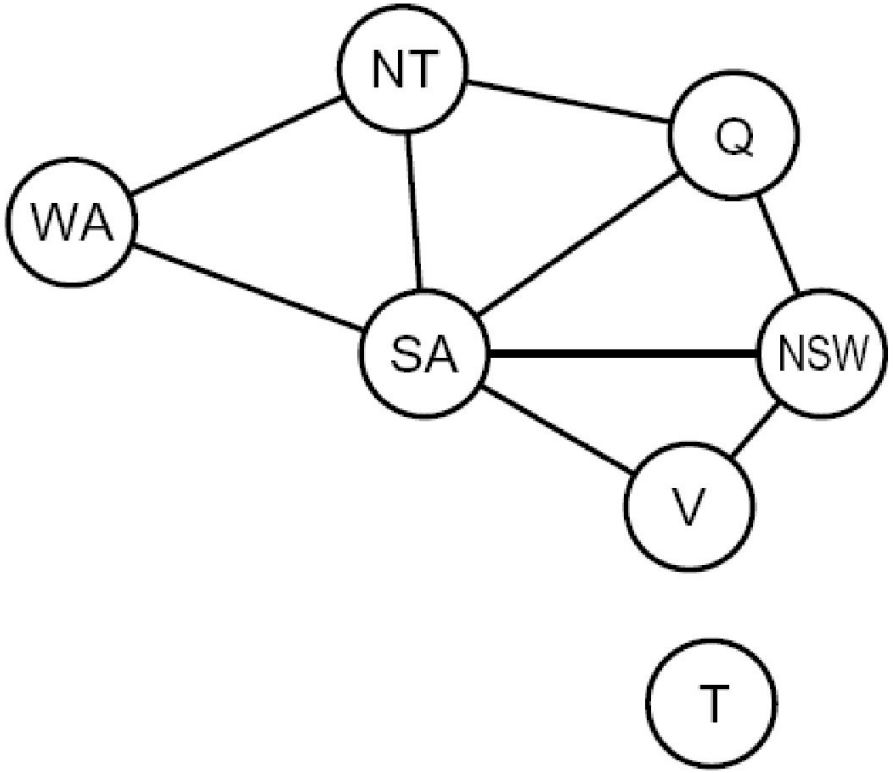
- Initial state: the empty assignment, {}
- Successor function: assign a value to an unassigned variable
- Goal test: the current assignment is complete and satisfies all constraints



Search Methods

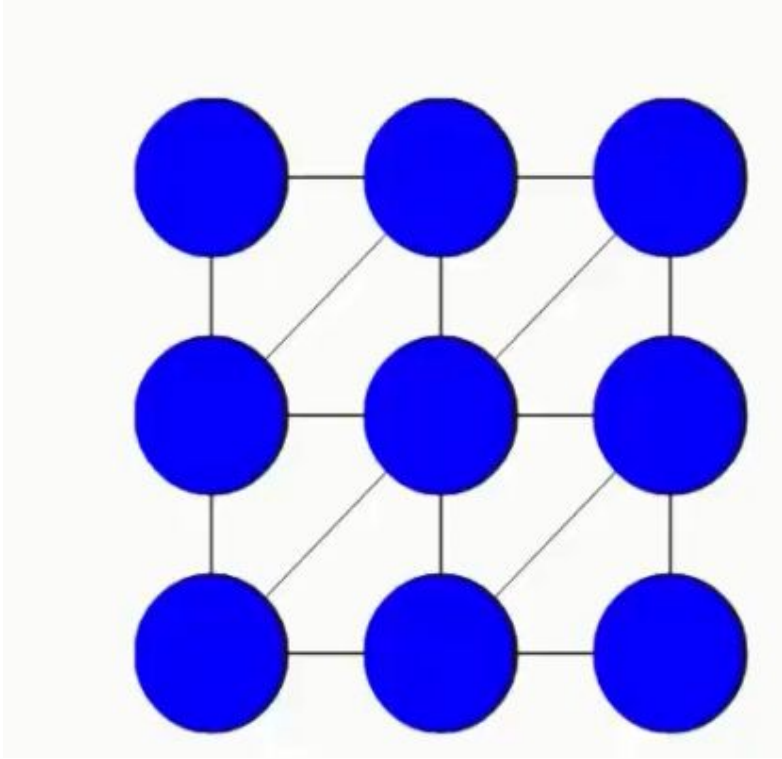
- What would BFS do?
  root ({}) has n * d successors
  all solutions at depth n

- What would DFS do?
  go down to depth n even though failure happens much earlier



[Demo: coloring -- dfs]

Demo Coloring -- DFS



Search Methods

- What problems does naïve search have and how can we solve them?