9/15/09 ① Constraint Satisfaction I

9:55 First: Where are we?

- We've finished the "Search" section, i.e. we've learned many ways to define problem spaces, and different algorithms to find solns in those prob spaces
  - Uninformed Search: BFS, DFS, UCS, DLS, ID DFS, BS
  - Informed Search: Heuristics, Greedy BFS, A*, RBFS IDA*, SMA*
  - Local Search: HC, stochastic HC, SA, LBS, GA
  - Did not cover local search in cont. spaces, but will later

- How would you use these methods in your own area? Discussion

10:08 Main idea from before: problem solving is defined by searching in a space of states. | Standard search prob.

Constraint Satisfaction Problem (CSP)

CSP problems are defined by the constraints.

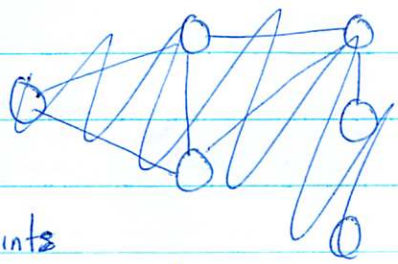
- set of variables X_i
- each variable can be assigned values V_j
- constraint

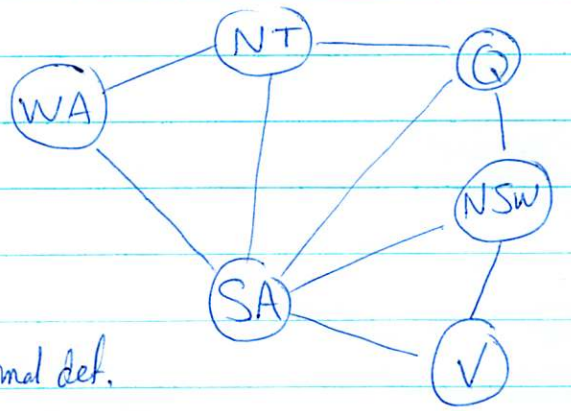
| variables
| domain of possible values
| constraint

Example Map Coloring:

Problem: Color each region with R, G, or B such that no neighboring region has the same color

How do we solve this?
First how do we represent the problem?





Contraints
No two connected nodes can have same color → (*) Formal def.

postpone

Could solve by 4 to local search methods

- initial state : no variable assignments
- Successor fn : assign value to any unassigned variable provided it doesn't conflict
- Goal test : have all variables been assigned?
- ~~no path cost~~

T

CS1 ③ [?] What are the properties of the soln?

- all variables assigned
- satisfies all constraints ← given by successor func.

⇒ Solns have depth n

② does path matter? No only interested in soln.

How many variable assignments? ~~#values~~

$$(\#\text{values})^{\#\text{variables}} = d^n$$

Formal definition of CSPs

$V = \{V_1, V_2, \ldots, V_n\}$ Values

$D = \{D_1, D_2, \ldots, D_d\}$ Domain

$C = \{\{\text{pair}\}, \{\text{legal values}\}\}$

$\{(WA, NT), \{(R,G), (R,B), (G,R), (G,B), (B,R), (B,G)\}\}$

represent by function, not explicitly

A CSP is a triplet $\{V, D, C\}$

→ General problem statement for large class of CSPs

CS1 ④ How do we solve this?

How about search.

- initial state: no vars assigned
- successor fn: What should this do?
  → Assign value to any unassigned var provided it satisfies constraints
- goal test: All vars assigned. Why?

Only interested in the soln, not the path.

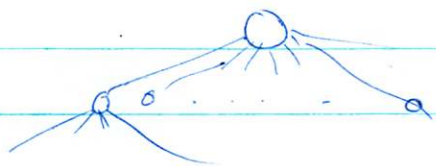
? what is the depth of the search tree?

depth n because there are n variables, when they are all assigned, we're done.

? How many possible assignments? Or, how big is the tree?

$(\#\text{vals})^{\#\text{vars}} = O(d^n)$ ⇒ still exponential in general

Which search method? BFS? What's the branching factor



each of n vars can be assign d vals
⇒ n×d not good.

next level (n-1)×d
⇒ $n! d^n$ for whole tree
Obviously bad. Only d^n possible assignments

Why does this happen? commutativity
order of "actions" doesn't matter.
In tree we reach many of the same partial assignments

Therefore:
generate successors by considering only a single variable at each node.

SA = R, G, or B not SA = R & WA = blue

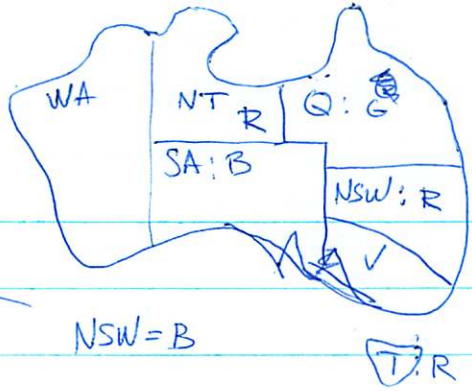
CSI (5) 10/12 What about DFS?

Called Backtracking Search:

- choose vals for one var at a time
- back track when var has no legal vals left to assign

```
func RBSearch (csp) returns soln or failure
    return RB({}, csp)
      α = assignment
func RB (α, csp) returns soln or failure
    if α complete return α
    var ← selectUnassignedVar (Vars(csp), α, csp)
    for each val in OrderDomainVals (var, α, csp) do   | picks legal values
        add (var = val) to α
        result ← RB (α, csp)
        if result ≠ failure then return result
        remove (var = val) from α
    return failure
```

CS1 (6) Backtracking success



{ }

foreach α = NSW = R | NSW = G | NSW = B

foreach NT = R | NT = G | NT = B — α = NSW = R , NT = R

foreach Q = G | Q = B — $\alpha = \alpha +$ Q = G

SA = B

T = R | T = G | T = B

V = G

WA = G

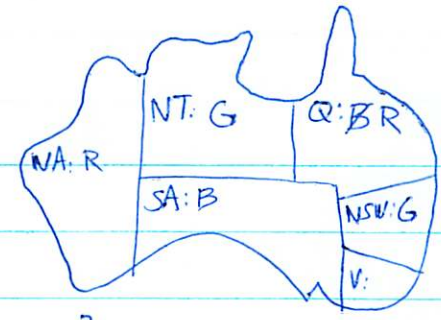
⋮

CSI (7)

foreach

$\alpha = \{WA = R\}$ $\{\}$

WA = R WA = G WA = B



foreach NT = G NT = B $\alpha = \{WA = R, NT = G\}$

remove ↓

foreach ~~Q = B~~ Q = ~~B~~ R $\alpha = \{WA = R, NT = G, \cancel{Q = B}, Q = R\}$

back track ← SA: {} → Whoops! No legal choices

SA = B $\alpha = \alpha + \{SA = B\}$

⋮

~~NSW~~ NSW = G

V = R

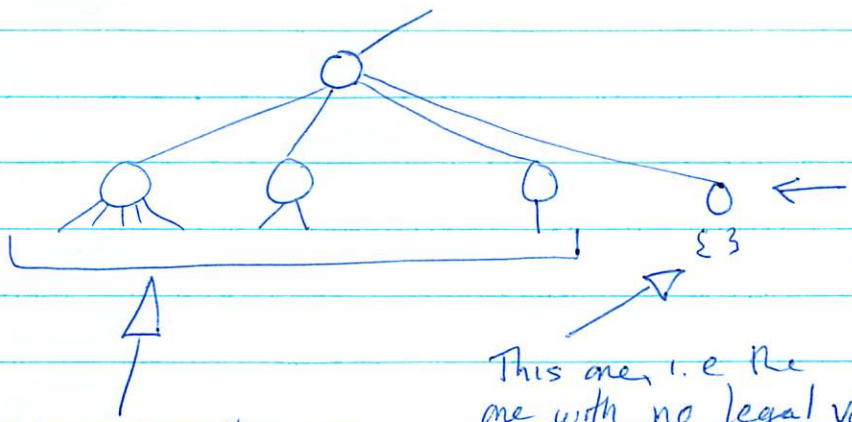
~~T~~ | T = R T = G T = B

CS1 (8)

In Backtracking Search, we had choices of
1) What variable should be assigned next
2) What order to consider the values

Do these choices matter? Can they improve search?

Suppose you're deep in the tree:



← unassigned vars
which order makes the most sense?

This one, i.e the one with no legal values because you detect failure immediately.

Otherwise, all the work here is wasted.

Minimum remaining values (MRV) heuristic:
- choose var with fewest "legal" values

aka • most constrained var
• fail-first

RN: 3-3000 times better than simple backtracking

? What about the first node? MRV doesn't help.
~~[redacted]~~ which would make the most sense?
SA because it constrains 5 other variables
This results in the greatest reduction of the search tree size
Called the **degree heuristic**