Constraint Satisfaction Problems (CSPs)

Set of variables V_i that can take on values from their Domains D_i (discrete / continuous)

Goal test specifies constraints on variables

- unary (on a particular variable)
  - e.g. left-most digit in crypt arithmetic is 0
- binary (on a pair of variables)
  - e.g. N-queens problem, map coloring
  - allowable cross-product of domains can be enumerated in discrete case
- other (on more than 2 variables)
  - e.g. many scheduling & layout problems

CSPs …

Depth of search tree = # variables

- Can use depth-first search

Regularly $b = \sum_i |Di|$,

but in CSPs, order of assignments does not matter, so we can assign one variable at a time, starting from V_1

- $b_i = |Di|$

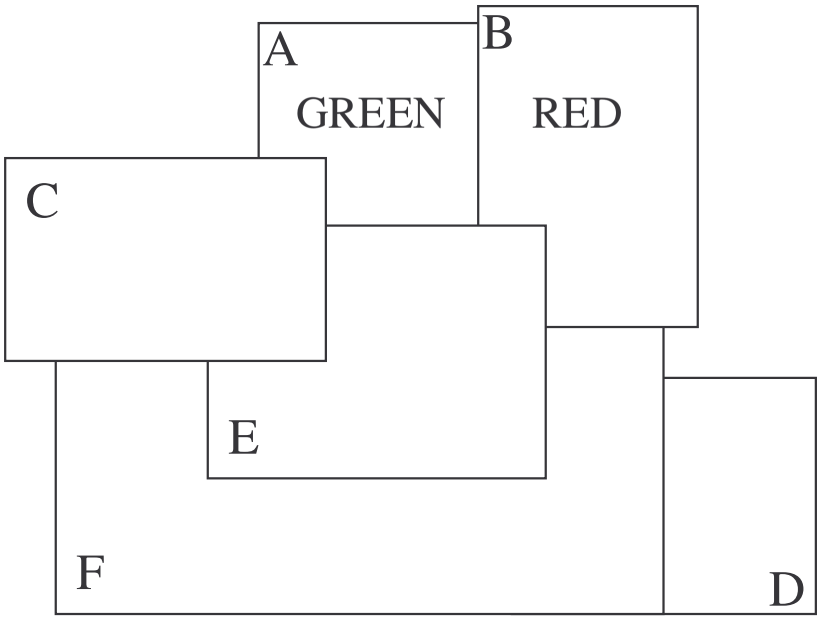
Example:

D_{shirt} = {BLUE, GREEN}

D_{pants} = {RED, YELLOW}

Example: Map coloring

$\forall_i \quad D_i = \{ \text{RED, GREEN, BLUE} \}$



A map-coloring problem after the first two variables (A and B) have been assigned values

Which variable (country) to choose next?

If we choose C, which value should we pick?

Using problem structure in CSPs

Straightforward depth-first-search will not always be efficient

use problem structure to improve (even this seems to not *always* be effective: CSPs include as subcases known NP-complete problems (e.g. 3SAT))

1. Backtracking: Prune the path if constraints violated

2. Forward checking: Each time a variable is instantiated, delete from the domains of uninstantiated variables all values that conflict with variables assigned so far. Backtrack if some domain becomes empty. E.g. if queens on six columns attack all squares on eighth column.

3. Arc consistency checking (forward checking is a special case). A state is arc consistent if every variable has a value (or none) that is consistent with the constraints. Successive elimination of values from domains. As preprocessing or during search.

Using problem structure in CSPs…

4. Most-constrained-variable heuristic:
   Choose a variable with fewest remaining possible values to instantiate next
   - Reduces the branching factor
   - Solves "bottleneck" variables early

5. Most-constraining-variable heuristic:
   Choose the variable that participates in the largest number of constraints on un-instantiated variables
   - Reduces the branching factor.

6. Least-constraining-value heuristic:
   Choose a value that rules out the smallest number of values in variables connected to the current variable by constraints

   CSPs can alternatively be solved using iterative improvement search methods