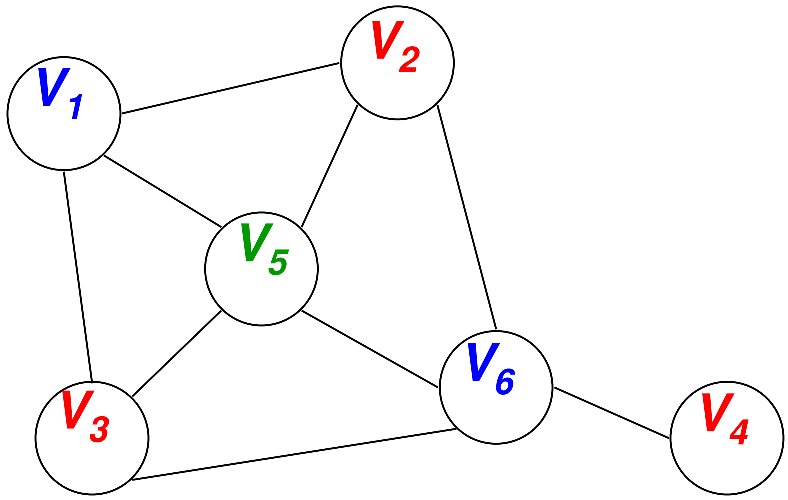
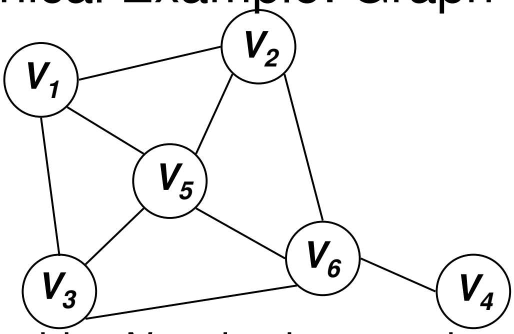
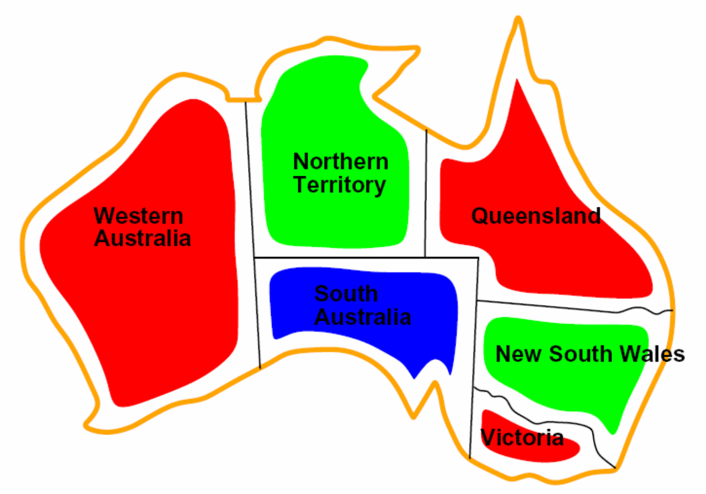


Canonical Example: Graph Coloring



- Consider *N* nodes in a graph
- Assign values $V_1,..,V_N$ to each of the *N* nodes
- The values are taken in {*R*,*G*,*B*}
- Constraints: If there is an edge between *i* and *j*, then V_i must be different of V_j

Canonical Example: Graph Coloring



CSP Definition

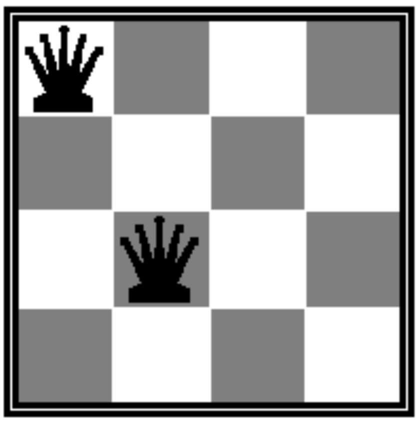
- CSP = {V, D, C}
- *Variables*: $V = \{V_1,..,V_N\}$
  - Example: The values of the nodes in the graph
- *Domain*: The set of d values that each variable can take
  - Example: $D = \{R, G, B\}$
- *Constraints*: $C = \{C_1,..,C_K\}$
- Each constraint consists of a tuple of variables and a list of values that the tuple is allowed to take for this problem
  - Example: $[(V_2,V_3),\{(R,B),(R,G),(B,R),(B,G),(G,R),(G,B)\}]$
- Constraints are usually defined implicitly → A function is defined to test if a tuple of variables satisfies the constraint
  - Example: $V_i \neq V_j$ for every edge (i,j)

Binary CSP

- Variable ***V*** and ***V'*** are connected if they appear in a constraint
- Neighbors of ***V*** = variables that are connected to ***V***
- The domain of ***V***, *D*(***V***), is the set of candidate values for variable ***V***
- $D_i = D(V_i)$

- Constraint graph for binary CSP problem:
  - Nodes are variables
  - Links represent the constraints
  - Same as our canonical graph-coloring problem

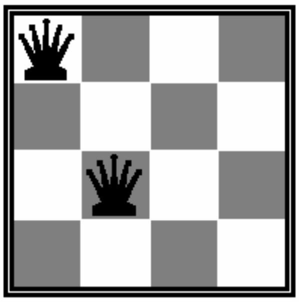
N-Queens



$Q_1 = 1 \quad Q_2 = 3$

Example: N-Queens

- Variables: Q_i
- Domains: $D_i = \{1, 2, 3, 4\}$
- Constraints
  - $Q_i \neq Q_j$ (cannot be in same row)
  - $|Q_i - Q_j| \neq |i - j|$ (or same diagonal)



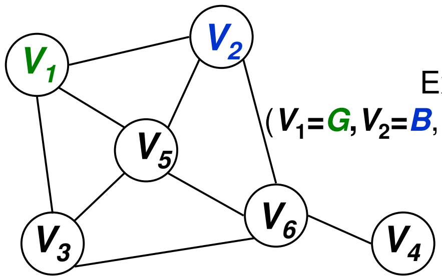
$Q_1 = 1 \quad Q_2 = 3$

- Valid values for (Q_1, Q_2) are (1,3) (1,4) (2,4) (3,1) (4,1) (4,2)

Cryptarithmetic

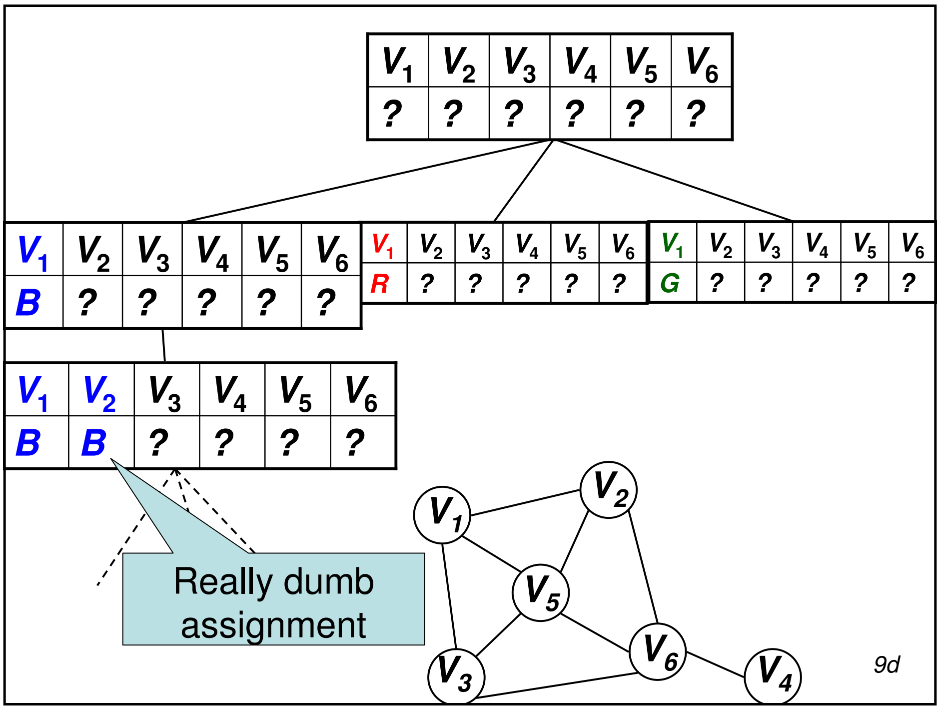
```
  S E N D
+ M O R E
---------
M O N E Y
```

Search Space

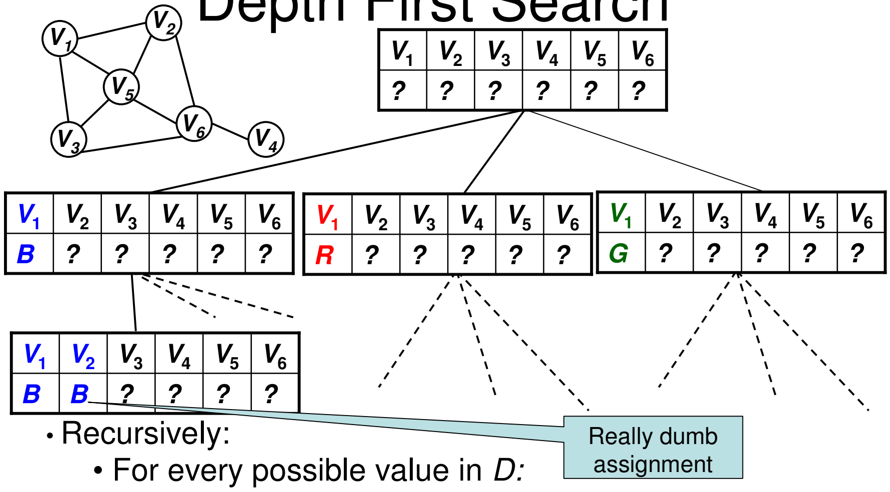


Example state:
$(V_1=G, V_2=B, V_3=?, V_4=?, V_5=?, V_6=?)$

- *State*: assignment to *k* variables with *k*+1,..,*N* unassigned
- *Successor*: The successor of a state is obtained by assigning a value to variable *k*+1, keeping the others unchanged
- *Start state*: $(V_1=?, V_2=?, V_3=?, V_4=?, V_5=?, V_6=?)$
- *Goal state*: All variables assigned with constraints satisfied
- No concept of cost on transition → We just want to find a solution, we don't worry how we get there



Depth First Search



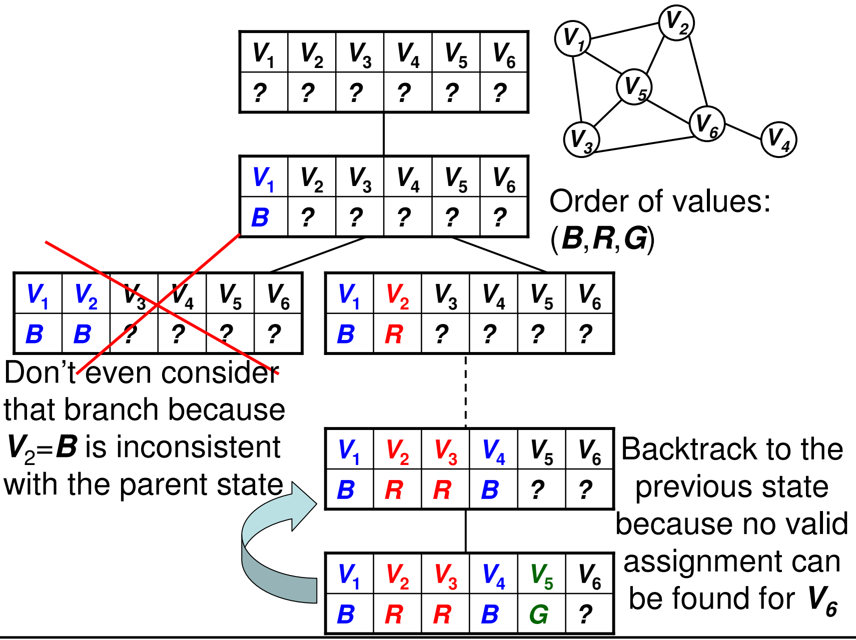
- Recursively:
  - For every possible value in *D:*
    - Set the next unassigned variable in the successor to that value
    - Evaluate the successor of the current state with this variable assignment
    - Stop as soon as a solution is found

9d

DFS

- Improvements:
  - Evaluate only value assignments that do not violate any constraints with the current assignments
  - Don't search branches that obviously cannot lead to a solution
  - Predict valid assignments ahead
  - Control order of variables and values

Backtracking DFS



Backtracking DFS

- For every possible value *x* in *D:*
  - If assigning *x* to the next unassigned variable V_{k+1} does not violate any constraint with the *k* already assigned variables:
    - Set the variable V_{k+1} to *x*
    - Evaluate the successors of the current state with this variable assignment
- If no valid assignment is found: Backtrack to previous state
- Stop as soon as a solution is found

9b, 27b

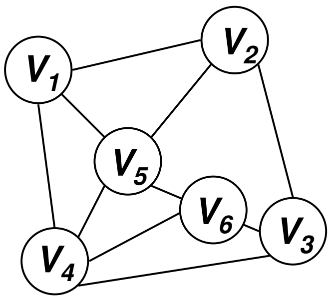
Backtracking DFS Comments

- Additional computation: At each step, we need to evaluate the constraints associated with the current candidate assignment (variable, value).
- Uninformed search, we can improve by predicting:
  - What is the effect of assigning a variable on all of the other variables?
  - Which variable should be assigned next and in which order should the values be evaluated?
  - When a branch fails, how can we avoid repeating the same mistake?

Forward Checking

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

	V_1	V_2	V_3	V_4	V_5	V_6
R	?	?	?	?	?	?
B	?	?	?	?	?	?
G	?	?	?	?	?	?

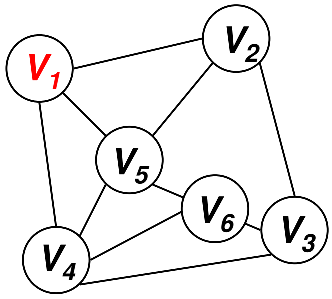


Warning: Different example with order (R,B,G)

Forward Checking

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

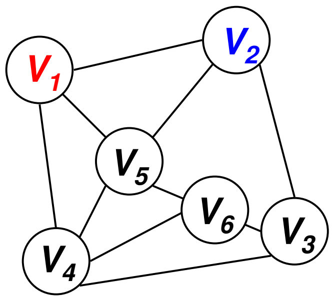
	V_1	V_2	V_3	V_4	V_5	V_6
R	O	X	?	X	X	?
B		?	?	?	?	?
G		?	?	?	?	?



Forward Checking

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

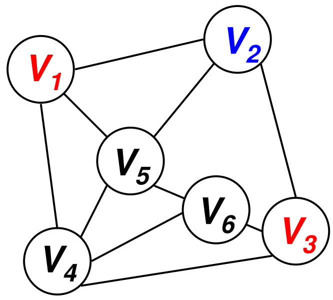
	V_1	V_2	V_3	V_4	V_5	V_6
R	O		?	X	X	?
B		O	X	?	X	?
G			?	?	?	?



Forward Checking

- Keep track of remaining legal values for unassigned variables
- Backtrack when no variable has a legal value

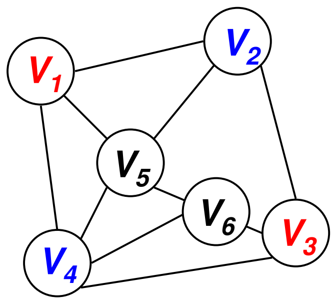
	V_1	V_2	V_3	V_4	V_5	V_6
R	O		O	X	X	X
B		O		?	X	?
G				?	?	?



Forward Checking

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

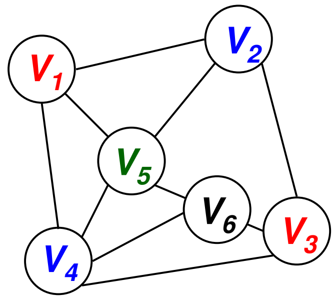
	V_1	V_2	V_3	V_4	V_5	V_6
R	O		O		X	X
B		O		O	X	X
G					?	?



Forward Checking

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

	V_1	V_2	V_3	V_4	V_5	V_6
R	***O***		***O***			***X***
B		***O***		***O***		***X***
G					***O***	***X***



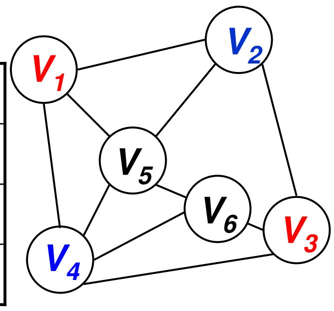
There are no valid assignments left for V_6 we need to backtrack

27f

Constraint Propagation

- Forward checking does not detect all the inconsistencies, only those that can be detected by looking at the constraints which contain the current variable.
- Can we look ahead further?

	V_1	V_2	V_3	V_4	V_5	V_6
R	***O***		***O***		***X***	***X***
B		***O***		***O***	***X***	***X***
G					***?***	***?***



At this point, it is already obvious that this branch will not lead to a solution because there are no consistent values in the remaining domain for V_5 and V_6.

Constraint Propagation

- ***V*** = variable being assigned at the current level of the search
- Set variable ***V*** to a value in *D*(***V***)
- For every variable ***V'*** connected to ***V***:
  - Remove the values in *D*(***V'***) that are inconsistent with the assigned variables
  - For every variable ***V''*** connected to ***V'***:
    - Remove the values in *D*(***V''***) that are no longer possible candidates
    - And do this again with the variables connected to ***V''***
      - ……..until no more values can be discarded

Constraint Propagation

New: Constraint Propagation

Forward Checking as before

- *V* = variable being assig…
- …value in …)
- …very variable ***V'*** conne…ted to ***V***:
  - …move the values in *D*(***V'***) that are inconsistent …h the assigned variables
  - …r every variable ***V''*** connected to ***V'***:
    - Remove the values in *D*(***V''***) that are no longer possible candidates
    - And do this again with the variables connected to ***V''***
      - ……..until no more values can be discarded