Constraint Satisfaction Problems

Deepak Kumar
October 2021

1

Search problems so far…

- Single agent
- Deterministic actions
- Fully observable state
- Discrete state space

2

1. Planning Problem

Given a problem, find a sequence of actions to go from start to goal:

Search(problem) returns [$action_1, action_2, \dots, action_n$]

The state is a Black Box
(i.e. *atomic representation*)

- `problem.getStartState()`
- `problem.isGoalState()`
- `problem.getSuccessors()`

Path to goal is important

Paths can have costs/depths

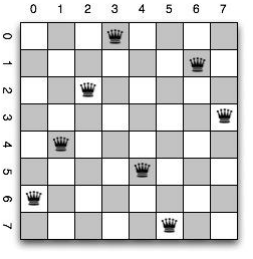
Heuristics help guide search

3

2. Identification Problem

- Assignments to variables (i.e. *factored representation*)
- Goal is important, not the path

N-queens - What is the placement of N queens so that all are safe?



Idea: use N variables representing N columns
Find an assignment from {0,1,…,N-1} to each variable.

4

Example 2: Map Coloring

- Given three colors: {red, green, blue}
- Color the map so that no two adjacent states have the same color.

5

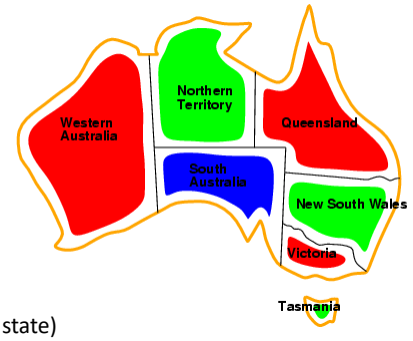
Example 2: Map Coloring

- Given three colors: {red, green, blue}
- Color the map so that no two adjacent states have the same color.

6

Example 2: Map Coloring

- Given three colors: {red, green, blue}
- Color the map so that no two adjacent states have the same color.



Idea: use 7 variables (one for each state)
Find an assignment of a color to each variable.

7

Example 3: Cryptarithmetic

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

Assign numbers in {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} to letters so it “adds up”.

8

Example 3: Cryptarithmetic

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

```
S = 9  M = 1
E = 5  O = 0
N = 6  R = 8
D = 7  E = 5
Y = 2
```

```
  9 5 6 7
+ 1 0 8 5
---------
1 0 6 5 2
```

9

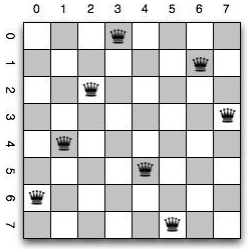
Example 4: Sudoku

4								5
	6	1		4			3	
			5		1		9	
		6		7		5		
	8		6	2	5		1	
		5		1		9		
	2		4		6			
	5			9		3	4	
7								6

Idea: use 81 variables (one for each state)
Find an assignment of a number from {1,2,…,9} to each variable.

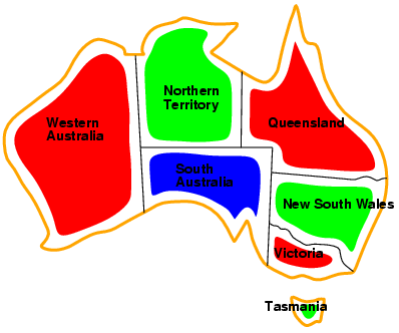
10

Constraint Satisfaction Problems!!



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

4								5
	6	1		4			3	
			5		1		9	
		6		7		5		
	8		6	2	5		1	
		5		1		9		
	2		4		6			
	5			9		3	4	
7								6



11

CSP Formulation (as a special case of search)

- State is defined by n variables

$$\{x_1, x_2, \ldots, x_n\}$$

- Variables can take on values from a domain set (One for each variable)

$$\{D_1, D_2, \ldots, D_n\}$$

- Goal test is a set of constraints specifying allowable combinations of values of variables (subsets)
- This allows general purpose algorithms without resorting to domain specific heuristics.

12

Example Formulation: N-Queens V1

- **Variables:** N^2 variables x_{ij} one for each square
- **Domains:** all variables have same domain $\{0, 1\}$ 0 - no queen
1 - queen
- **Constraints**

1. In a row, any two cells are either empty or exactly one has a queen
$$\forall_{i,j,k}(x_{ij}, x_{ik}) \in \{(0,0), (0,1), (1,0)\}$$
2. In a column, any two cells are either empty or exactly one has a queen
$$\forall_{i,j,k}(x_{ij}, x_{kj}) \in \{(0,0), (0,1), (1,0)\}$$
3. In a diagonal, any two cells are either empty or exactly one has a queen
$$\forall_{i,j,k}(x_{ij}, x_{i+k,j+k}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall_{i,j,k}(x_{ij}, x_{i-k,j-k}) \in \{(0,0), (0,1), (1,0)\}$$
4. There are exactly N queens
$$\sum_{i,j} x_{i,j} = N$$

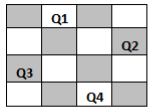
13

N-Queens V2

- Variables: $Q_1, Q_2, \dots, Q_N$
- Domains: $\{1, 2, 3, \dots, N\}$
- Constraints:
$$\forall_{i,j} \text{nonThreatening}(Q_i, Q_j)$$
- Solution (N=4): $\{Q_1 = 2, Q_2 = 4, Q_3 = 1, Q_4 = 3\}$

14

N-Queens V2



- Variables: $Q_1, Q_2, \dots, Q_N$
- Domains: $\{1, 2, 3, \dots, N\}$
- Constraints:

$$\forall_{i,j} \text{nonThreatening}(Q_i, Q_j)$$

- Solution (N=4): $\{Q_1 = 2, Q_2 = 4, Q_3 = 1, Q_4 = 3\}$

15

Example: Map-Coloring

- **Variables:** *WA, NT, Q, NSW, V, SA, T*
- **Domains:** $D_i = \{red, green, blue\}$
- **Constraints:** adjacent regions must have different colors

e.g., $WA \neq NT$

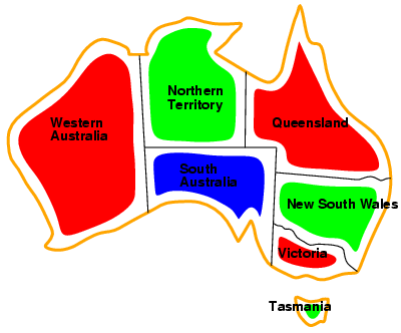
or
(WA, NT)
$\in \{(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)\}$

16

Example: Map-Coloring

- Solutions are complete and consistent assignments

$$\{WA = red,\\ NT = green,\\ Q = red,\\ NSW = green,\\ V = red,\\ SA = blue,\\ T = green\}$$

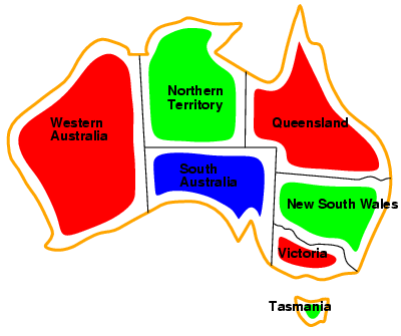


17

Example: Map-Coloring

- Solutions are complete and consistent assignments

$$\{WA = red,\\ NT = green,\\ Q = red,\\ NSW = green,\\ V = red,\\ SA = blue,\\ T = green\}$$

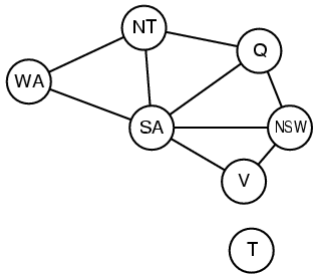


- **Complete** – Every variable has a value assigned.
- **Consistent** – The assignment satisfies all constraints.

18

Constraint Graph Representation of CSP

- Binary CSP: each constraint relates two variables
- Constraint graph: nodes are variables, edges/arcs are constraints



19

Start with a basic search algorithm…

Initial State: Empty assignment { }

Successor Function: assign a value to an unassigned variable

Goal Test: current assignment complete & consistent?

20

Start with a basic search algorithm…

Initial State: Empty assignment { }

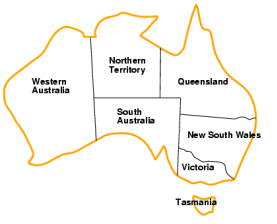
Successor Function: assign a value to an unassigned variable

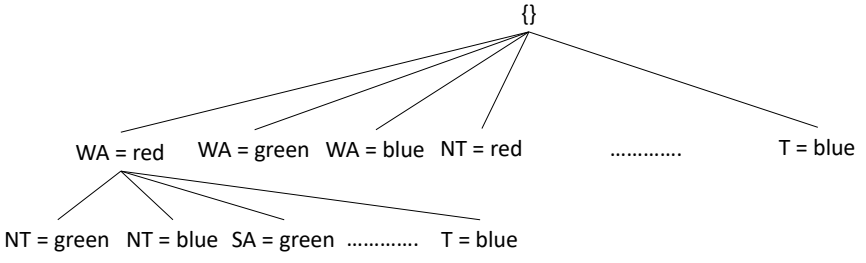
Goal Test: current assignment complete & consistent?

Which search algorithm to use??

21

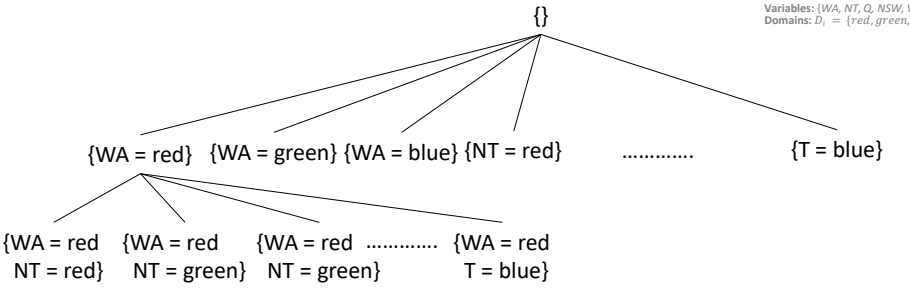
Try Breadth-First Search





22

Try Breadth-First Search



Branches

n*d (7*3=21)

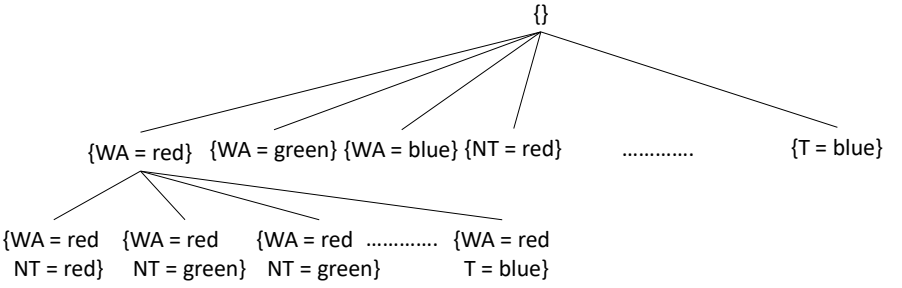
(n-1)*d (6*3=18)

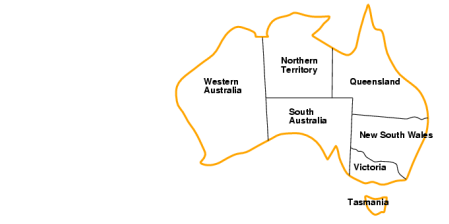
$n! * d^n$ $(6! * 3^7)$

But there are only d^n complete assignments!
Most are not consistent!!

23

Try Breadth-First Search





Branches

n*d (7*3=21)

(n-1)*d (6*3=18)

$n! * d^n$ $(6! * 3^7)$

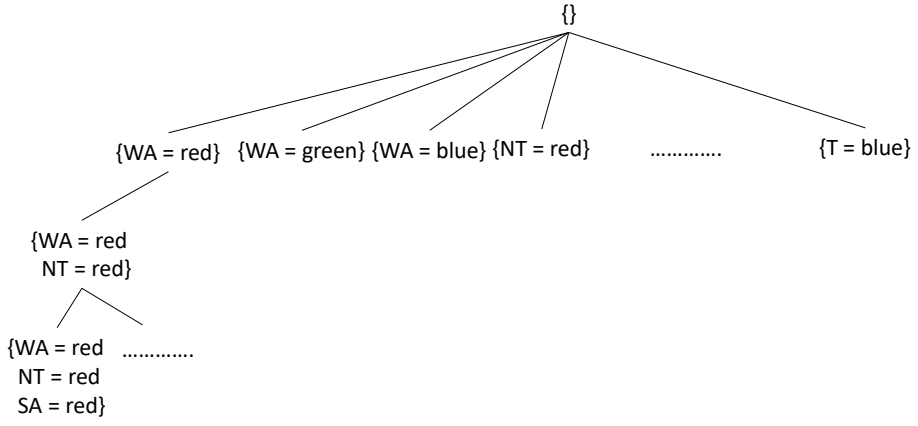
1,574,640

There are only d^n complete assignments!
Most are not consistent!!

2,187

24

What about Depth-First Search?



Branches

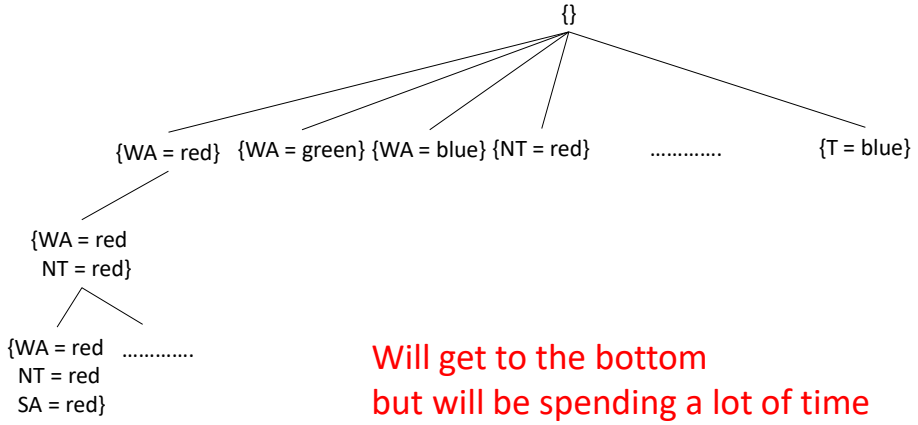
n*d (7*3=21)

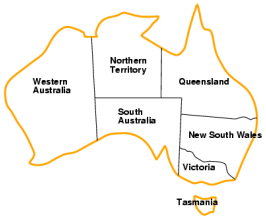
(n-1)*d (6*3=18)

$n! * d^n$ $(6! * 3^7)$

25

What about Depth-First Search?





Branches

n*d (7*3=21)

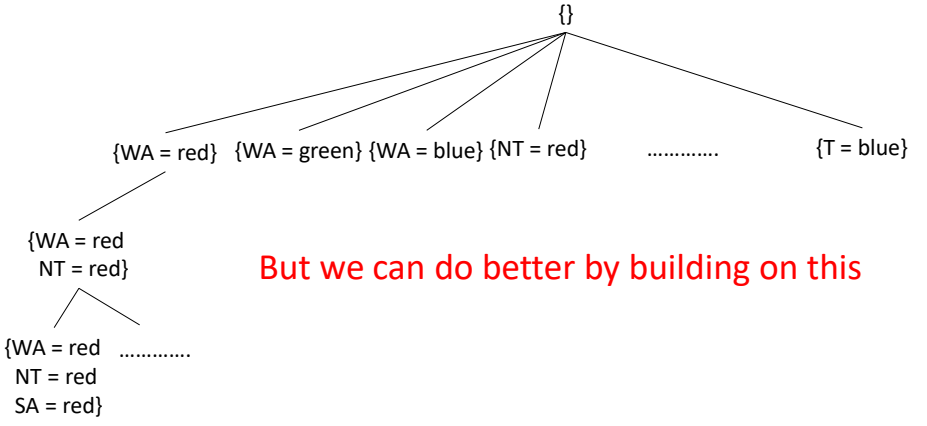
(n-1)*d (6*3=18)

$n! * d^n$ $(6! * 3^7)$

Will get to the bottom
but will be spending a lot of time
looking at failure options.

26

What about Depth-First Search?



But we can do better by building on this

Branches

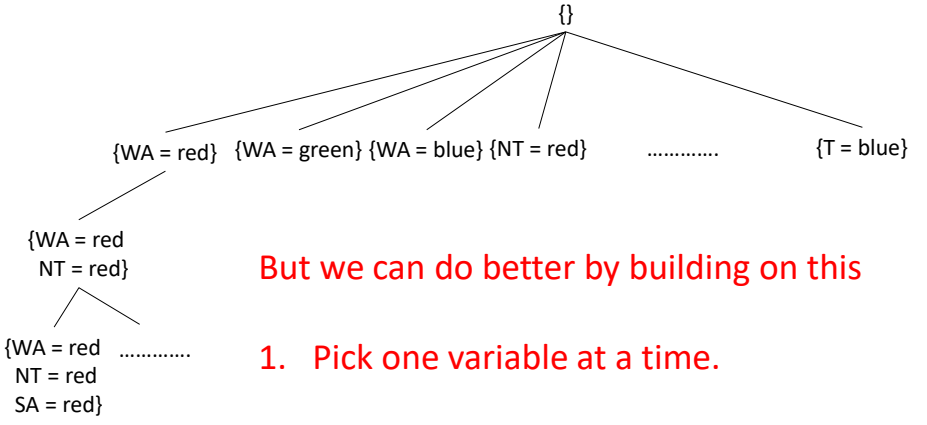
n*d (7*3=21)

(n-1)*d (6*3=18)

$n! * d^n$ $(6! * 3^7)$

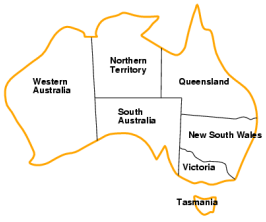
27

What about Depth-First Search?



But we can do better by building on this

1. Pick one variable at a time.
2. Check constraints as you go. (incremental goal testing)



Branches

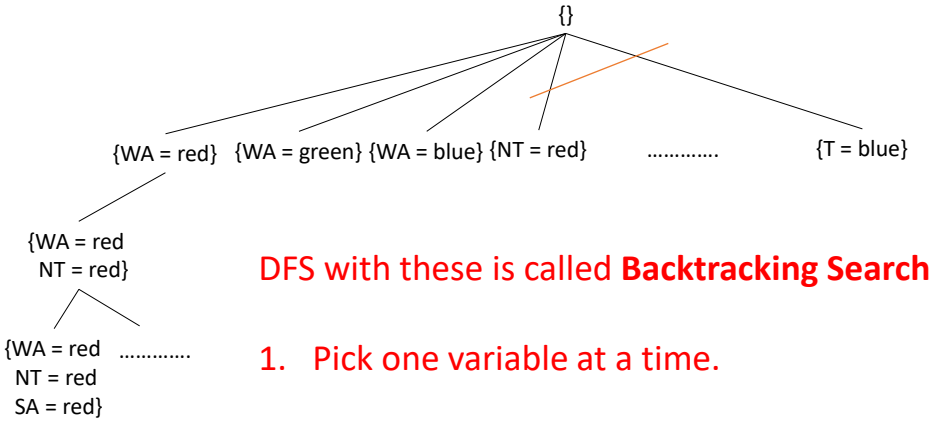
n*d (7*3=21)

(n-1)*d (6*3=18)

$n! * d^n$ $(6! * 3^7)$

28

What about Depth-First Search?



Branches

~~n*d~~ (3)

~~(n-1)*d~~ (3)

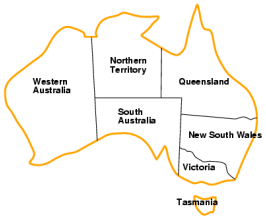
$\sout{n! * d^n}$ (3^7)

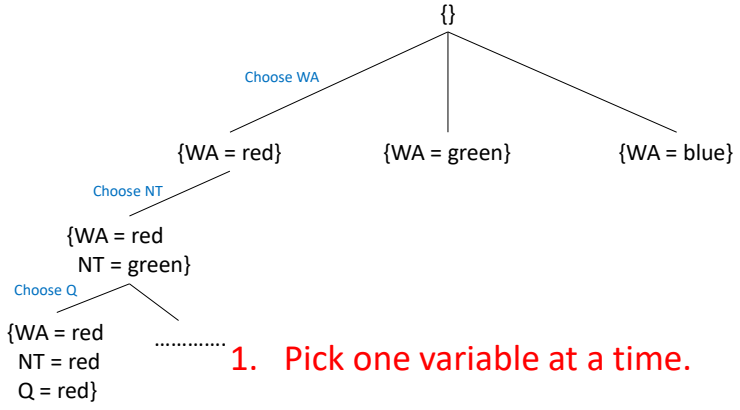
DFS with these is called **Backtracking Search**

1. Pick one variable at a time.
2. Check constraints as you go. (incremental goal testing)

29

Backtracking Search





1. Pick one variable at a time.
2. Check constraints as you go. (incremental goal testing)

30

Backtracking Search Algorithm

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

function BACKTRACK(csp, assignment) returns a solution or failure
  if assignment is complete then return assignment

  var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)

  for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
    add {var = value} to assignment
    inferences ← INFERENCE(csp, var, assignment)

    if inferences ≠ failure then
      add inferences to csp
      result ← BACKTRACK(csp, assignment)

      if inferences ≠ failure then return result
      remove inferences from csp

    remove {var = value} from assignment
  return failure
```

CSP
State: $\{x_1, x_2, \dots, x_n\}$
Domains: $\{D_1, D_2, \dots, D_n\}$
Constraints: …

assignment
$\{x_1 = v_1, x_2 = v_2, \dots, x_n = v_n\}$