Constrained Satisfaction Problem(CSP)

- Constraint Satisfaction Problems (CSP)
- Backtracking search for CSPs
- Local search for CSPs

Constrained Satisfaction Problem (CSP)

Idea:

We have a number of variables, each with a set of possible values, and we have to assign a value to each variable so that the constraints of the problem are satisfied.

A constraint-satisfaction problem (CSP) consists of

- Variables
- Set of allowed values (for each variable)
- Constraints

Constrained Satisfaction Problem (CSP)

- In basic search problems, we are explicitly given a goal (or perhaps a small set of goals) and need to find a path to it.
- In a constraint-satisfaction problem, we're given only a *description of what constraints a goal state must satisfy.* Target is to find a *goal state that satisfies these constraints*.

Constrained Satisfaction Problem (CSP)

- Assumptions about the world: a single agent, deterministic actions, fully observed state, discrete state space

Planning: sequences of actions

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

Identification: assignments to variables

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

**CSPs are a type of identification problem*

Constrained Satisfaction Problem (CSP)

A CSP consists of three main components:

X : a set of **variables** $\{X_1,\ldots,X_n\}$

D : a set of **domains** $\{D_1,\ldots,D_n\}$, one domain per variable
(*i.e. the domain of X_i is D_i , which means that X_i can only be assigned values from D_i* . Here, we're only going to consider **finite domains**.

C : is a set of **constraints** that specify allowable assignments of values to variables.

(for example, a binary constraint consists of a pair of different variables, (X_i,X_j) , and a set of pairs of values that X_i and X_j can take on at the same time)

Constrained Satisfaction Problem (CSP)

Example 1: suppose we have a CSP as follows:

- Three variables X_1 , X_2 and X_3
- Domains: $D_1 = \{1,2,3,4\}$, $D_2 = \{2,3,4\}$, and $D_3 = \{3,7\}$

  X_1 can only be assigned one of the values 1,2,3 or 4
- Constrained : No pair of variables have the same value, i.e. $X_1 \neq X_2$, $X_1 \neq X_3$, and $X_2 \neq X_3$

we can explicitly describe each of these constraints as a relation between the two variables where the pairs show the allowed values that the variables can be simultaneously assigned, i.e.

$X_1 \neq X_2 = \{(1,2),(1,3),(1,4),(2,3),(2,4),(3,2),(3,4),(4,2),(4,3)\}$
$X_1 \neq X_3 = \{(1,3),(1,7),(2,3),(2,7),(3,7),(4,3),(4,7)\}$
$X_2 \neq X_3 = \{(2,3),(2,7),(3,7),(4,3),(4,7)\}$

Constrained Satisfaction Problem (CSP)

Example 2: Suppose we have a CSP as follows:

- Three variables X_1 , X_2 and X_3
- Domains: $D_1 = \{1,2,3,4\}$, $D_2 = \{2,3,4\}$, and $D_3 = \{3,7\}$

  X_1 can only be assigned one of the values 1,2,3 or 4
- Constrained : $X_1 + X_2 = X_3$, and X_1 is even.

Here

$X_1 + X_2 = X_3$ is a ternary constrained

$X_1 + X_2 = X_3 = \{ (1,2,3), (3,4,7) , (4,3,7)\}$

X_1 is even

$X_1 = \{2,4\}$

The only one case where X_1 is even and satisfies both condition is (4,3,7). So the solution to the problem is $X_1 = 4$, $X_2 = 3$ and $X_3 = 7$

Constrained Satisfaction Problem (CSP)

- Variables can be assigned, or unassigned
- If a CSP has some variables assigned, and those assignments don't violate any constraints,
  we say the CSP is **consistent**, or that it is **satisfied**
- If all the variables of a CSP are assigned a value, we call that a **complete assignment** bb
- A **solution** to a CSP complete assignment that is consistent, i.e. a complete assignment that does not violate any constraints
- Depending on the constraints, a CSP may have 0, 1, or many solutions
  - A CSP with no solutions is sometimes referred to as **over-constrained**
  - A CSP with many solutions is sometimes referred to as **under-constrained**
- If a domain is empty, then the CSP is no solution

CSP Examples: Map Coloring

The map coloring problem requires that we color regions of the map such that neighboring regions never have the same color.

Variables : WA, NT, SA, NSW, Q, V, T

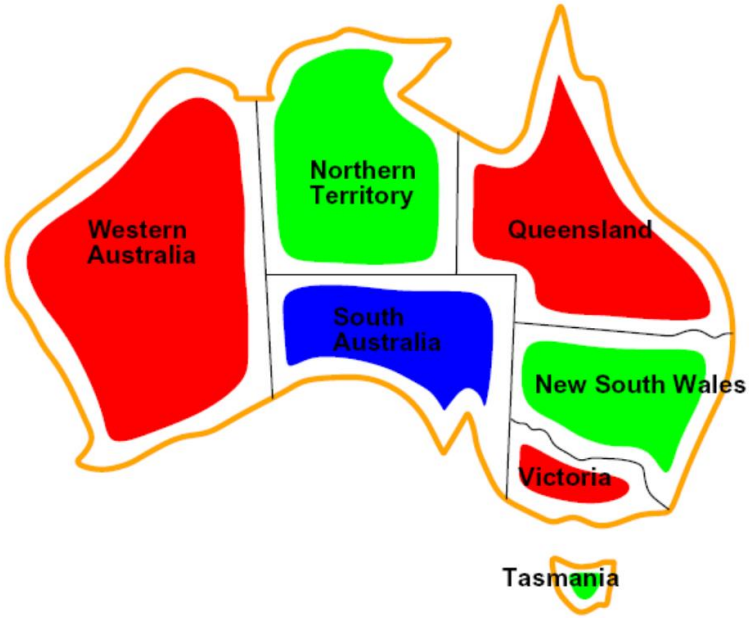
Domains : { RED, GREEN , BLUE }

Constraints: adjacent regions must have different colors

Solutions are assignments satisfying all constraints, e.g.

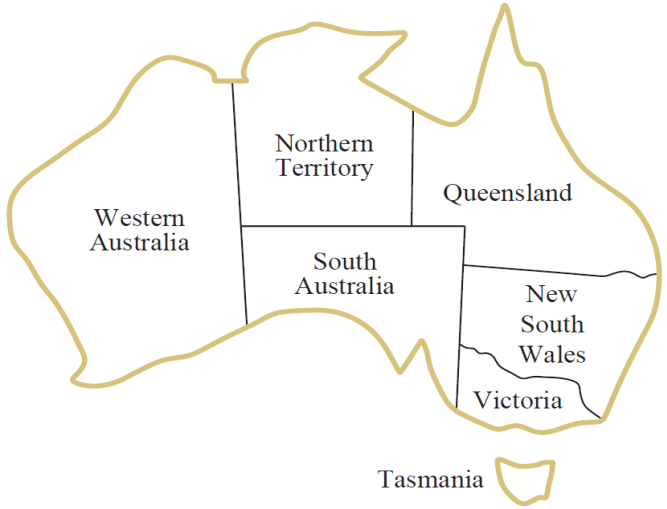
{ WA= RED, NT= GREEN, SA=BLUE,

Q=RED, NSW=GREEN, V=RED, T=GREEN }

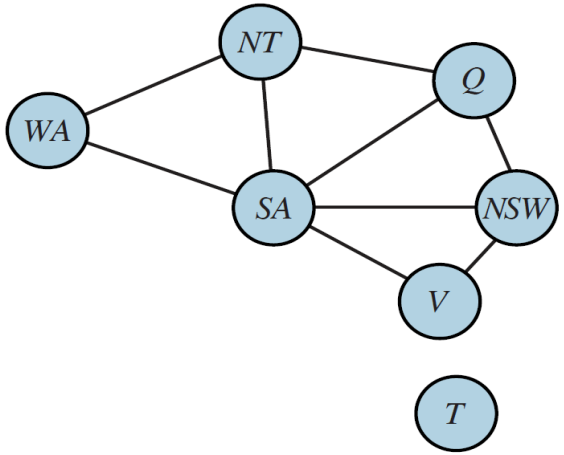


CSP Examples: Map Coloring

- It can be helpful to visualize a CSP as a constraint graph, as shown in Figure (b). The nodes of the graph correspond to variables of the problem, and an edge connects any two variables that participate in a constraint.
- *Constraint={ WA ≠ NT, WA ≠ SA, SA ≠ NT, SA ≠ Q, SA ≠ NSW, SA ≠ V, NT ≠ Q, Q ≠ NSW, NSW ≠ V }*



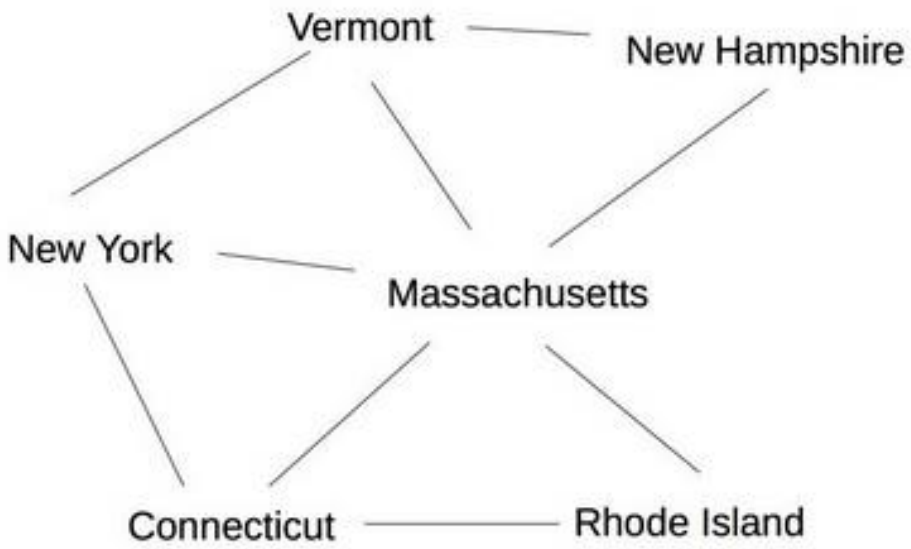
(a)



(b)

CSP Examples: Map Coloring

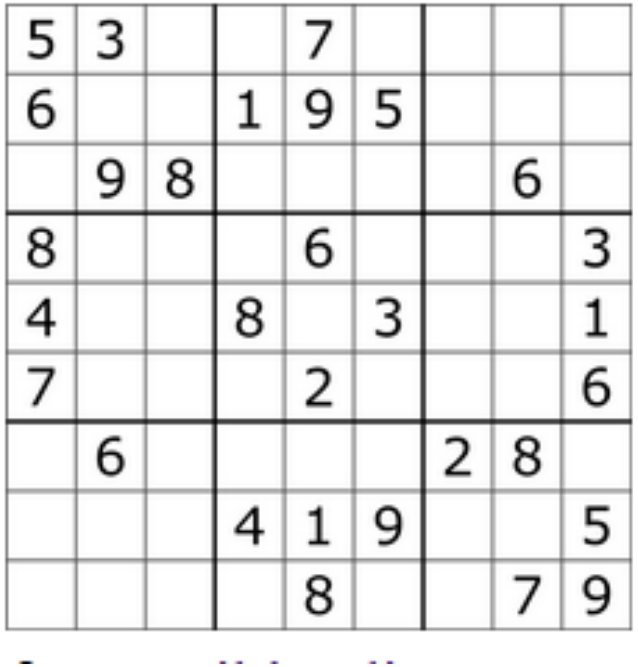
How many colours in the domains required so that no two neighbours have same color



Note : that graph coloring is NP complete

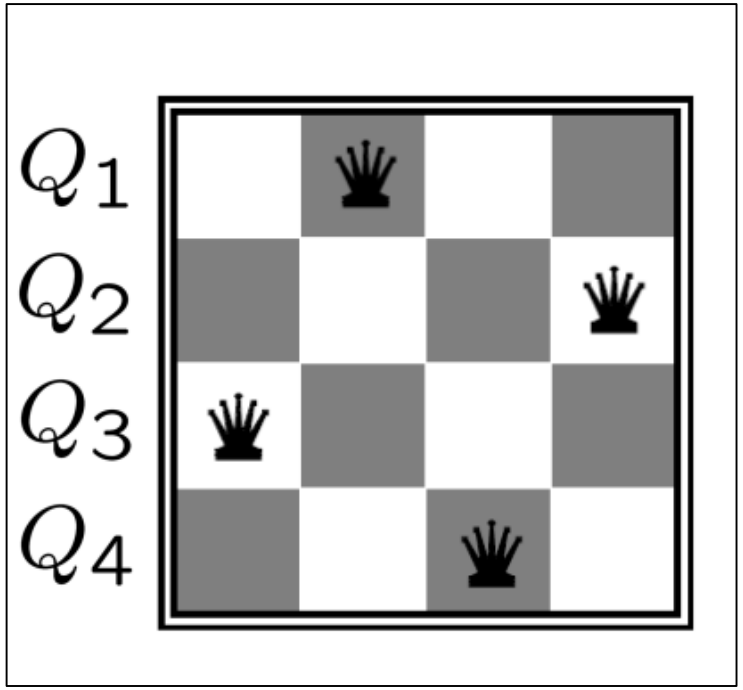
CSP Examples: Sudoku

Sudoku can be modelled as a CSP problem,
Variables: cells x_{ij}
Values: {1,2,...,9}
Constraints: no duplicates in row,
no duplicates in column, no duplicates in block.



CSP Examples: N-queen

Remember that a queen in chess threatens another piece if there is a direct path between the two either horizontally, or vertically, or diagonally.
In the **n queens problem,** we have to place n queens on a n by n board so that no queen threatens another.



Variables: Q_k
Domains: {1,2,3,…N}
Constrained: For all *i,j* non-threatening (Q_i , Q_j)
(Q_i , Q_j) ∈ { (1 , 3), (1, 4)….}

CSP Examples: Cryptarithmetic puzzles

Each letter in a cryptarithmetic puzzle represents a different digit. For the case in Figure 6.2(a), this would be represented as the global constraint *Alldiff (F, T, U, W, R, O)*

Variable: F T U W R O C1 C2 C3

Domains: { 0, 1 , 2, 3, 4, 5, 6, 7, 8, 9}

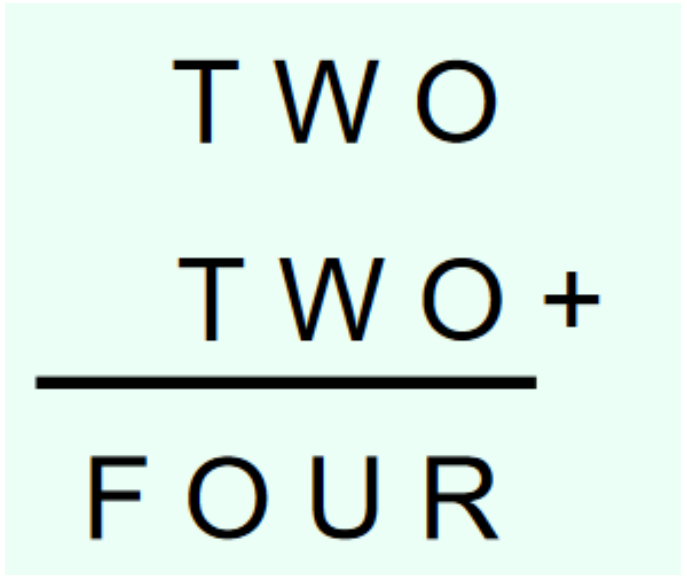
Constraints: all variable values should be different.

O + O = R + 10 * C1

C1 + W + W = U + 10 * C2

C2 + T + T = O + 10 * C3

C3 = F



where C1, C2, and C3 are auxiliary variables representing the digit carried over into the tens, hundreds, or thousands column.

CSP Examples: Cryptarithmetic puzzles

The constraint hypergraph for the cryptarithmetic problem, showing the *Alldiff* constraint (square box at the top) as well as the column addition constraints (four square boxes in the middle). The variables C1,C2, and C3 represent the carry digits for the three columns from right to left

O + O = R + 10 * C1

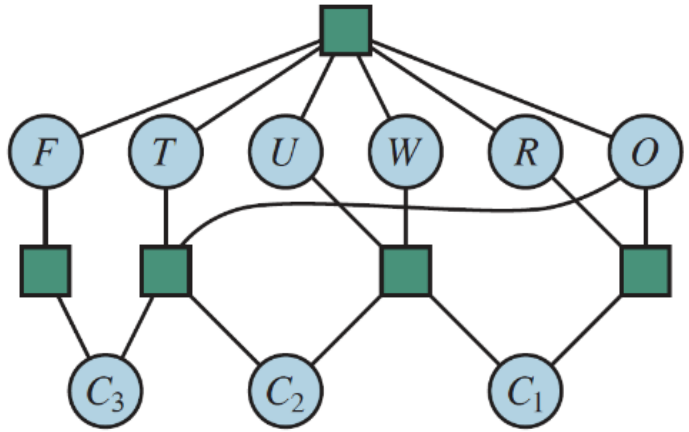
C1 + W + W = U + 10 * C2

C2 + T + T = O + 10 * C3

C3 = F

$$\begin{array}{cccc} & T & W & O \\ + & T & W & O \\ \hline F & O & U & R \end{array}$$

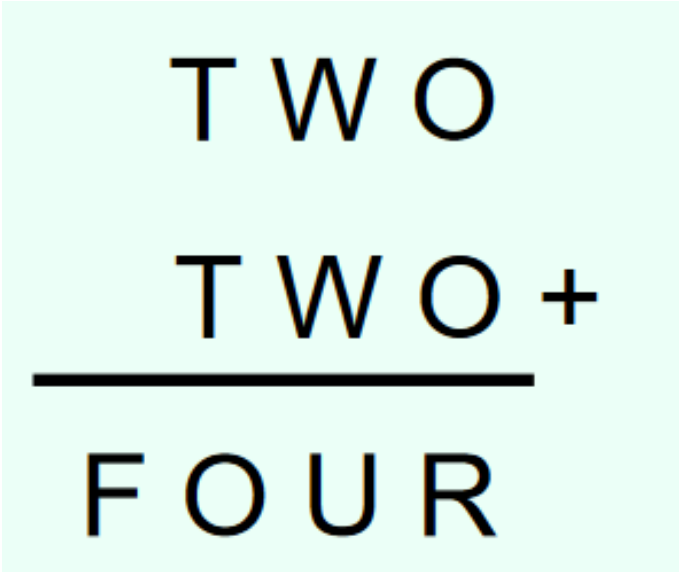
(a)



(b)

These constraints can be represented in a constraint hypergraph, such as the one shown in Figure b. A hypergraph consists of ordinary nodes (the circles in the figure) and hypernodes (the squares), which represent *n-ary* constraints—constraints involving n variables.

CSP Examples: Cryptarithmetic puzzles



Setting F = 1, O = 4, R = 8, T = 7, W = 3,
U = 6 gives 734 + 734 = 1468

CSP Examples: Cryptarithmetic puzzles

Constraints:

- A number 0-9 is assigned to a particular alphabet.
- Each different alphabet has a unique number.
- All the same, alphabets have the same numbers.
- The numbers should satisfy all the operations that any normal number does

$$\begin{array}{r} \text{T O} \\ +\ \text{G O} \\ \hline \text{O U T} \end{array}$$

$$\begin{array}{r} \boldsymbol{S\ E\ N\ D} \\ +\ \boldsymbol{M\ O\ R\ E} \\ \hline \boldsymbol{M\ O\ N\ E\ Y} \end{array}$$

CSP Examples: Cryptarithmetic puzzles

```
   T O
+  G O
-------
 O U T
```

```
   2 1
+  8 1
-------
 1 0 2
```

The problem:

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

A solution:

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

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
- Floor planning

Generate and Test

- *Generate and Test* is the simplest of the algorithms to identify a solution for a CSP.
- In this algorithm, each of the possible solutions is attempted (each value enumerated for each variable) and then tested for the solution.
- Since testing each combination of variable within the domain of each value can be extremely slow and inefficient, heuristics can be applied to avoid those combinations that are outside of the solution space

CSP as a standard search problem

- A CSP can easily be expressed as a standard search problem
- Incremental formulation

  Initial State: the empty assignment {}

  Successor function: Assign a value to any unassigned variable provided that it does not violate a constraint

  Goal test: the current assignment is complete and consistent

  Path cost: constant cost for every step (not generally relevant)

Commutativity

- CSPs are commutative.
- The order of any given set of actions has no effect on the outcome.

  Example: choose colors for Australian territories one at a time

  [WA=red then NT=green] same as [NT=green then WA=red]

- All CSP search algorithms can generate successors by considering assignments for only a single variable at each node in the search tree

Backtracking (CSP)

- The *backtracking* algorithm operates with a simple uninformed search algorithm, such as depth-first search.
- At each node, a variable is instantiated with a value and the constraint violations are checked.
- If the values are legal, search is permitted to continue, otherwise, the current branch is abandoned and the next node from the OPEN list is evaluated.

start state: no variables have a value

each move: assign a value to one variable

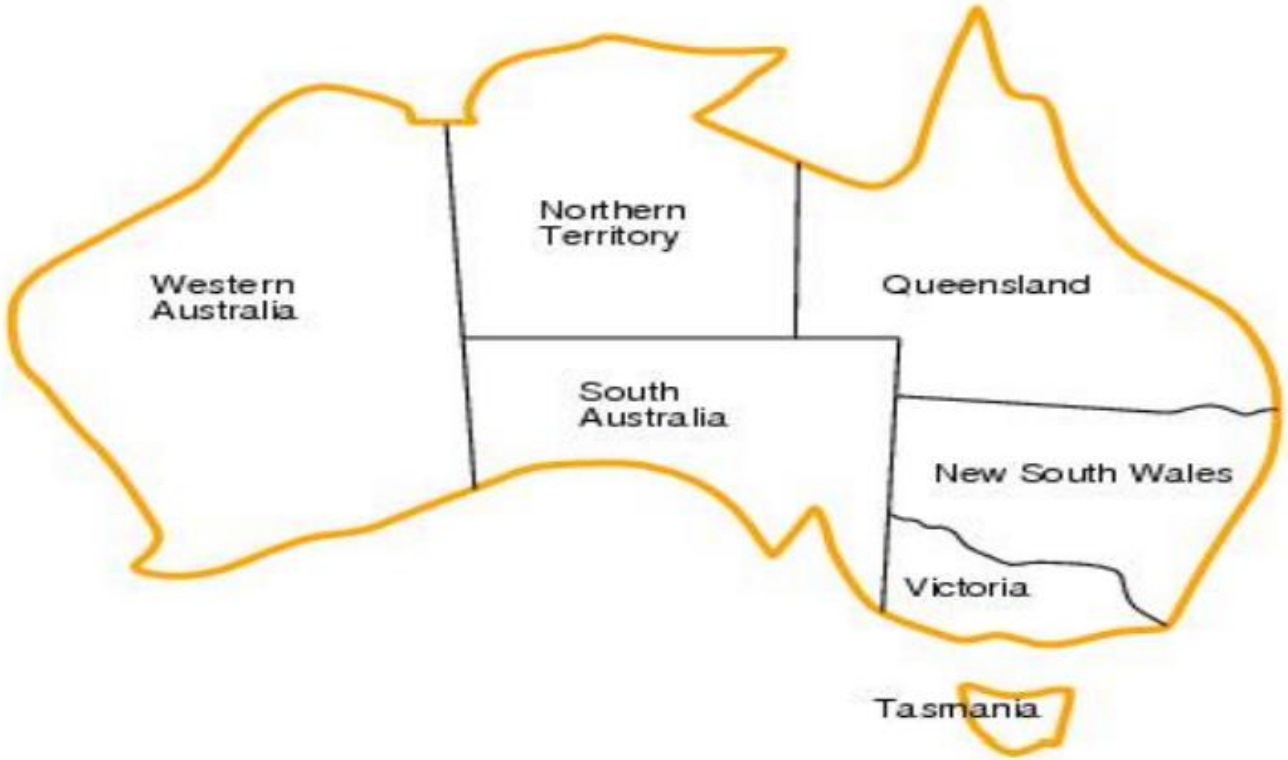
goal state: all variables have a value

Backtracking (CSP)

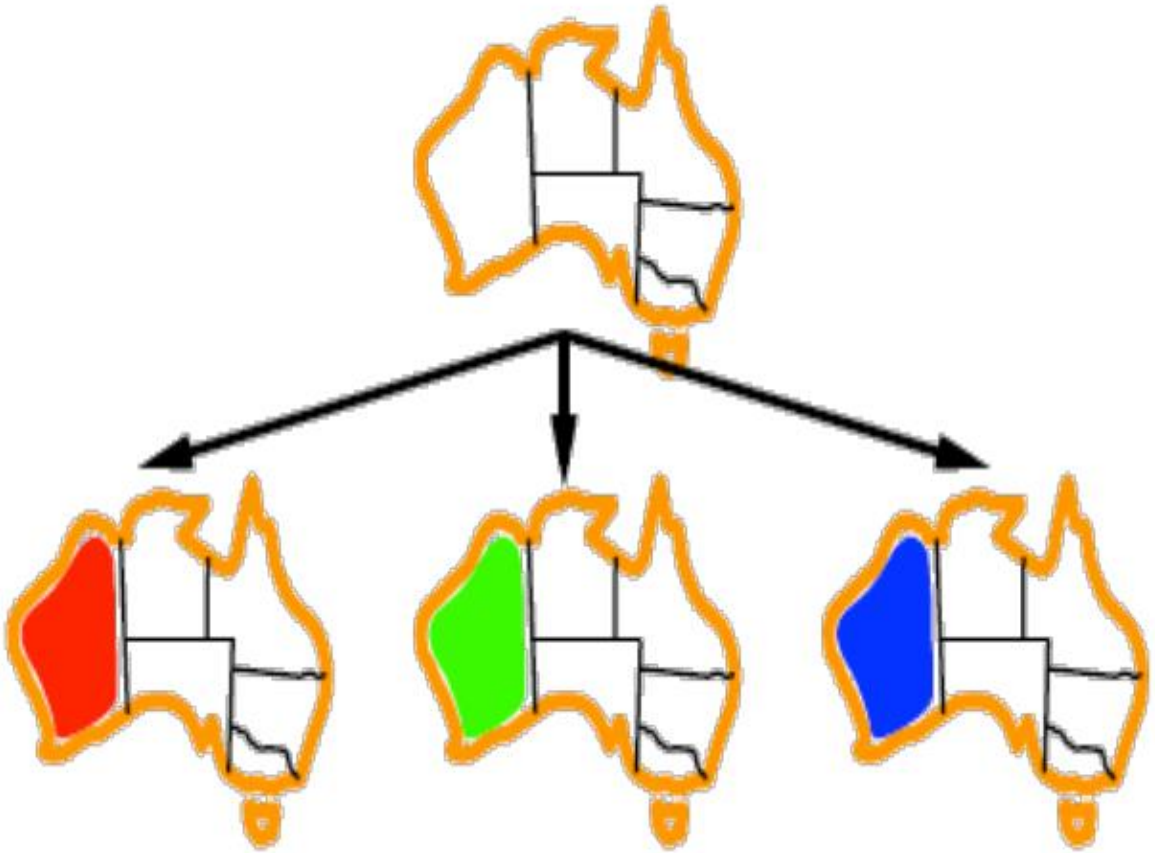
```
function BACKTRACKING-SEARCH(csp) return a solution or failure
    return RECURSIVE-BACKTRACKING({} , csp)

function RECURSIVE-BACKTRACKING(assignment, csp) return a solution or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp],assignment,csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
            if value is consistent with assignment according to CONSTRAINTS[csp] then
                    add {var=value} to assignment
                    result ← RRECURSIVE-BACTRACKING(assignment, csp)
                    if result ≠ failure  then return result
                    remove {var=value} from assignment
    return failure
```

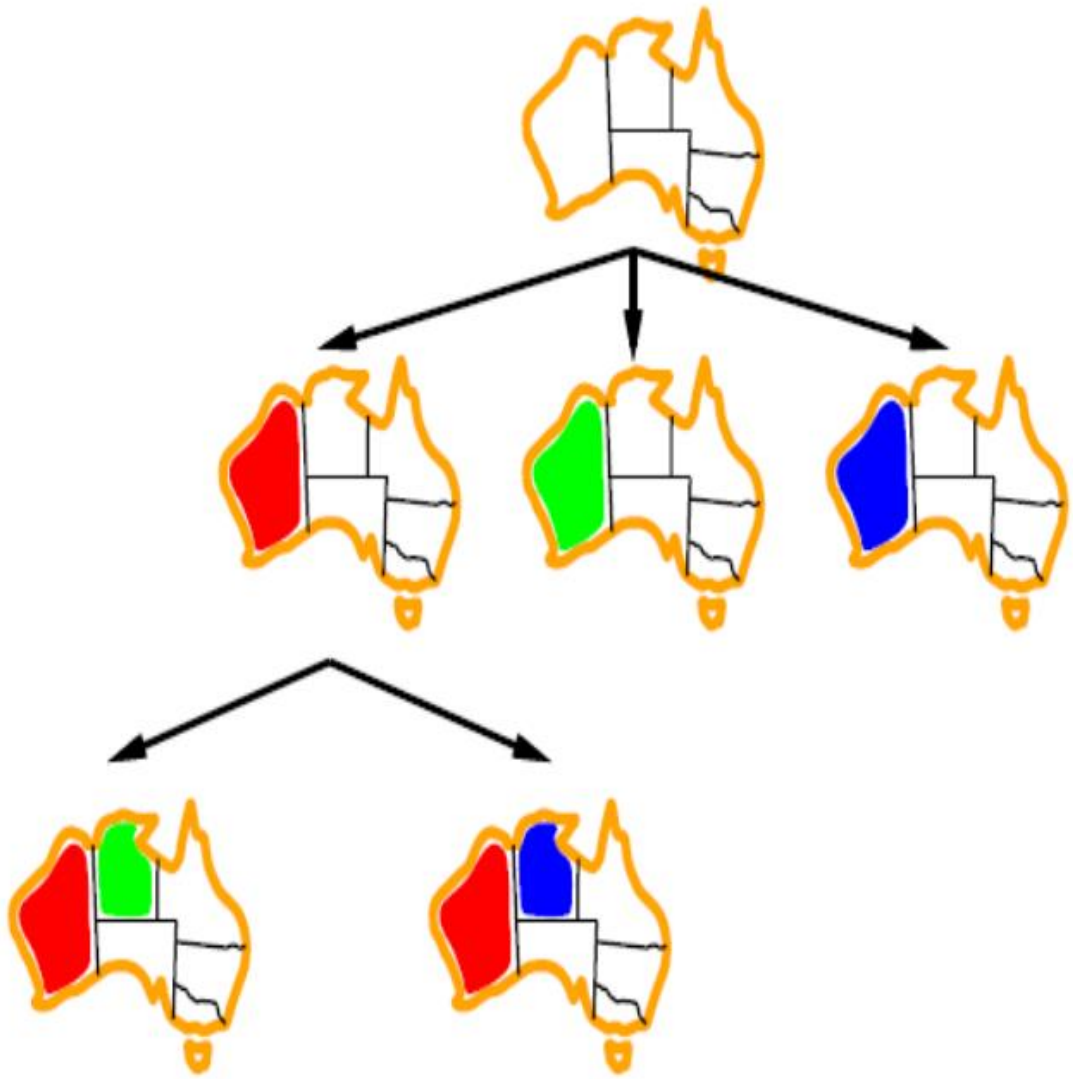
Backtracking example



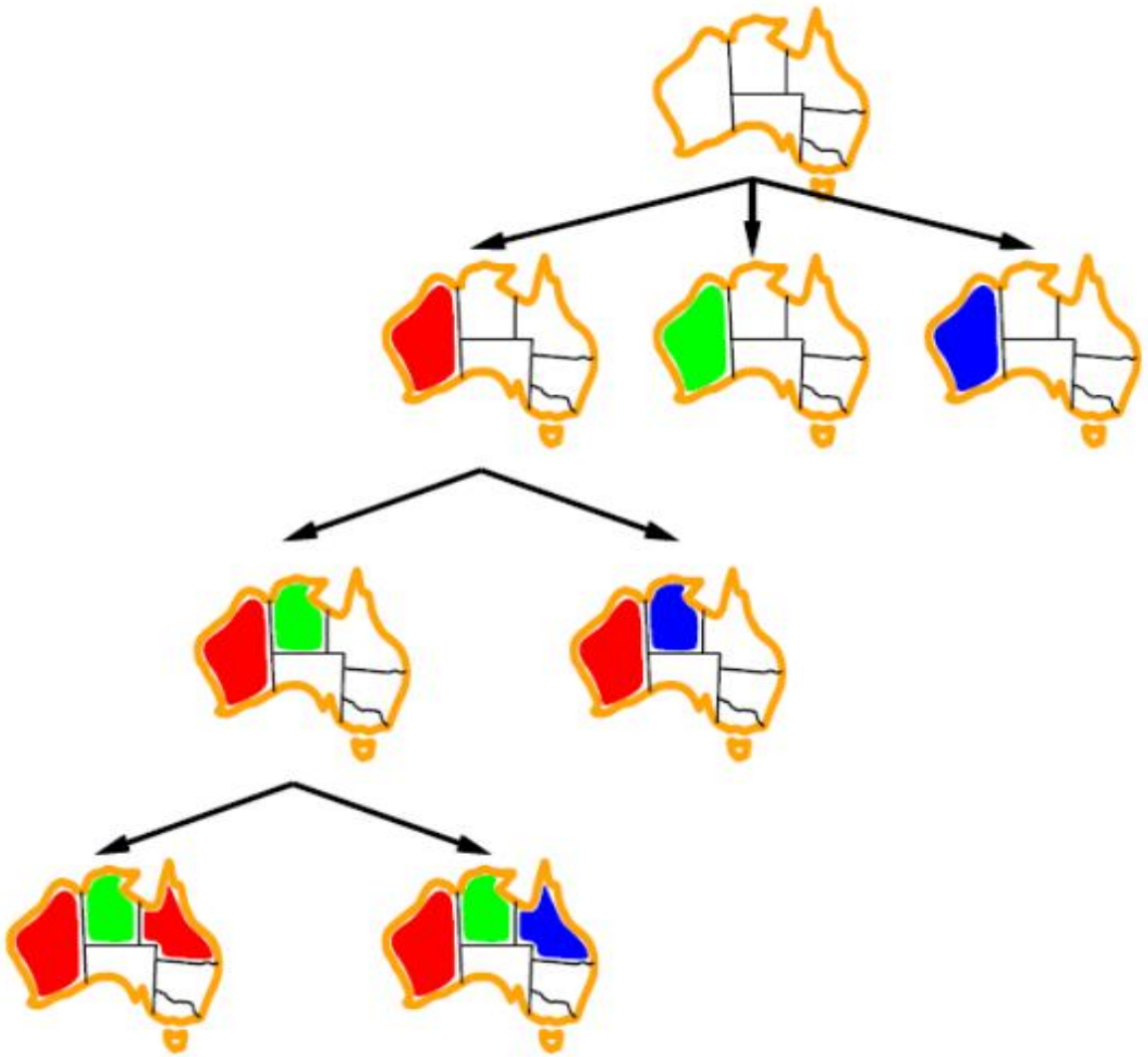
Backtracking example



Backtracking example



Backtracking example



Improving CSP efficiency

- Previous improvements on uninformed search

  → introduce heuristics

For CSPs, general-purpose methods can give large gains in speed,

→ Which variable should be assigned next?

→ In what order should its values be tried?

→ Can we detect inevitable failure early?

→ Can we take advantage of problem structure?

Heuristics for CSPs

CSP searches the space in the depth-first manner.

But we still can choose:

Which variable to assign next?

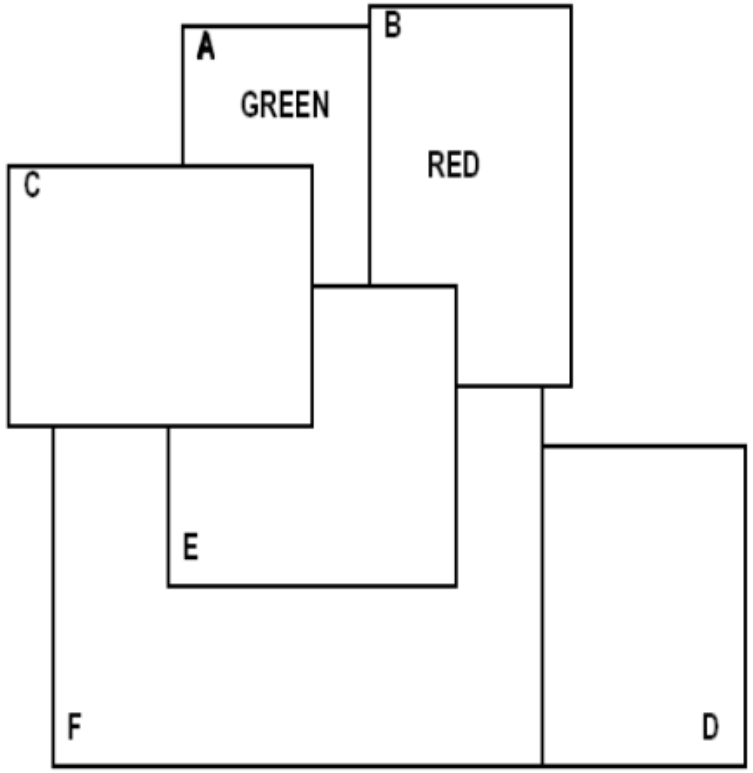
Which value to choose first?

Heuristics

- Most constrained variable
  - Which variable is likely to become a bottleneck?
- Least constraining value
  - Which value gives us more flexibility later?

Heuristics for CSPs

Examples: map coloring



Heuristics

- Most constrained variable

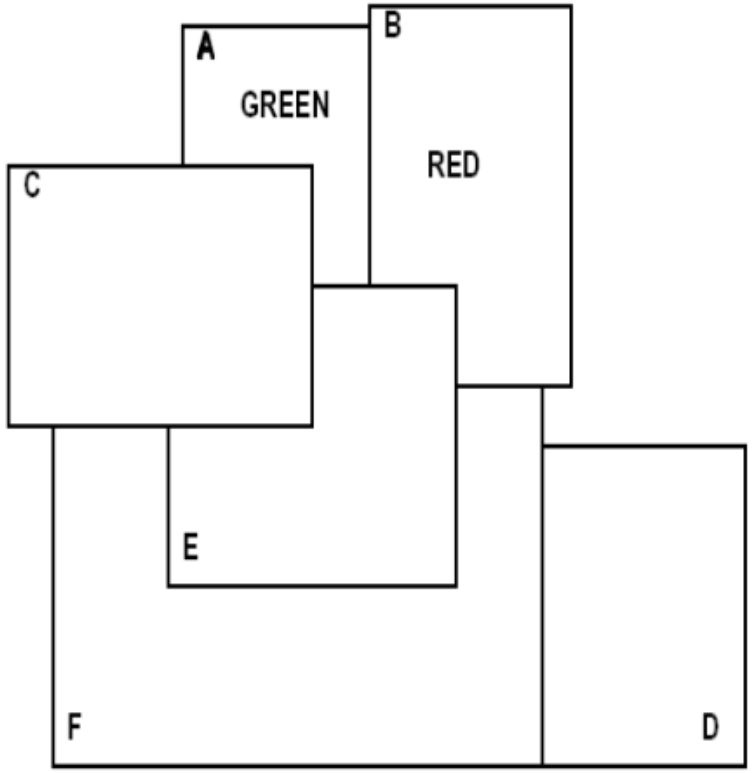
  ?

- Least constraining value

  ?

Heuristics for CSPs

Examples: map coloring



Heuristics

- Most constrained variable

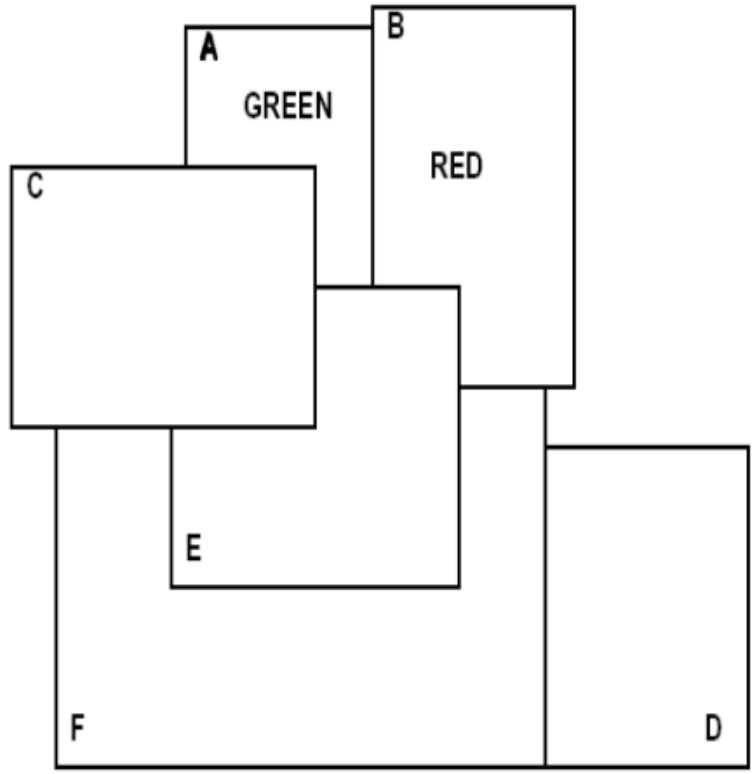
  Country E is the most constrained one (cannot use Red, Green)

- Least constraining value

  ?

Heuristics for CSPs

Examples: map coloring



Heuristics

- Most constrained variable

  Country E is the most constrained one (cannot use Red, Green)

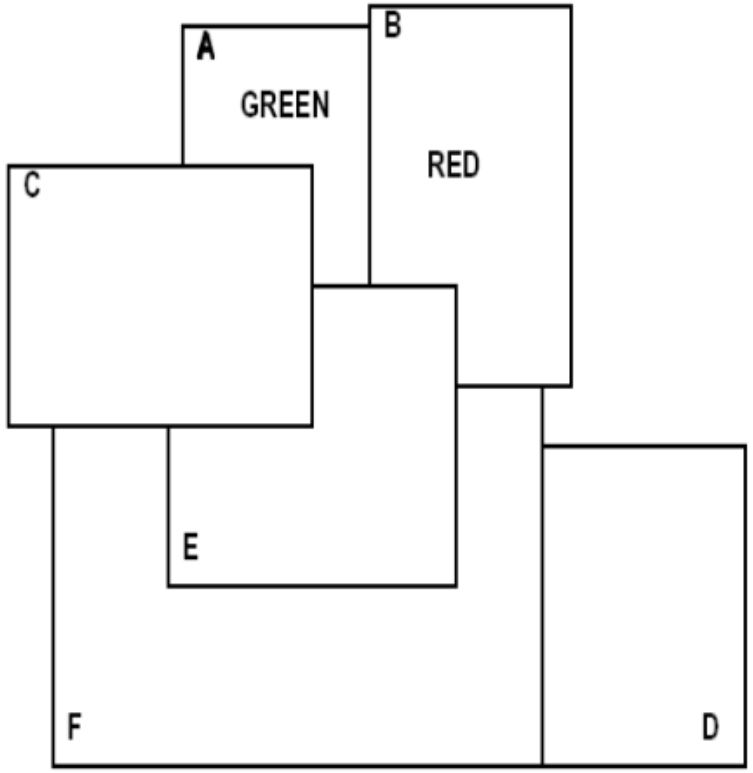
- Least constraining value

  -- Assume we have chosen variable C

  -- What color is the least constraining color?

Heuristics for CSPs

Examples: map coloring



Heuristics

- Most constrained variable

  Country E is the most constrained one (cannot use Red, Green)

- Least constraining value

  -- Assume we have chosen variable C

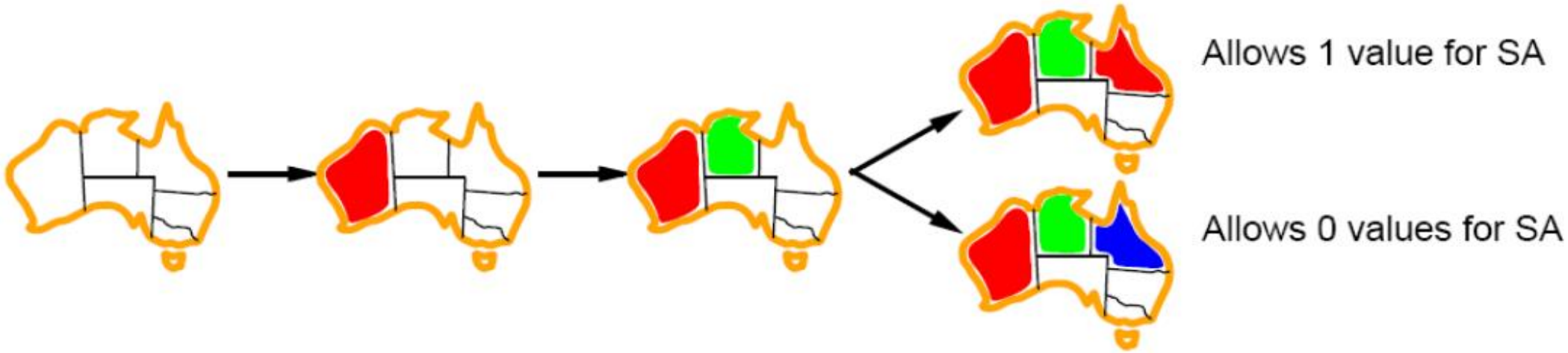
  -- Red is the least constraining valid color for the future

Minimum remaining values (MRV)

Most constrained variable Heuristic

Heuristic Rule: choose variable with the fewest legal moves
e.g., will immediately detect failure if X has no legal values

Least constraining value



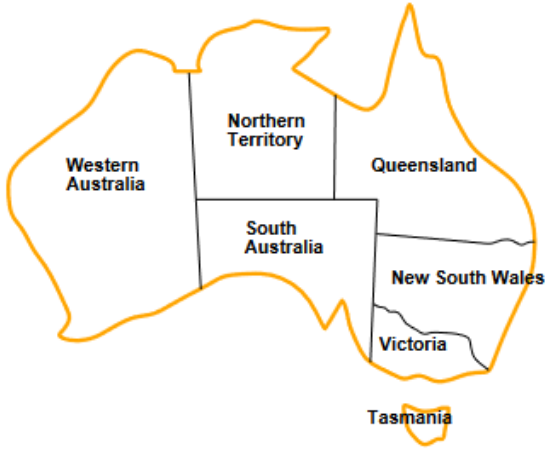
- Least constraining value heuristic
- Used to select order of values
- Heuristic Rule: given a variable choose the least constraining value

  leaves the maximum flexibility for subsequent variable assignments

Forward checking

Idea: Keep track of remaining legal values for unassigned variables

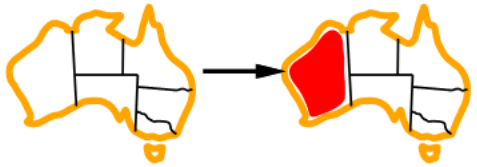
Terminate search when any variable has no legal values

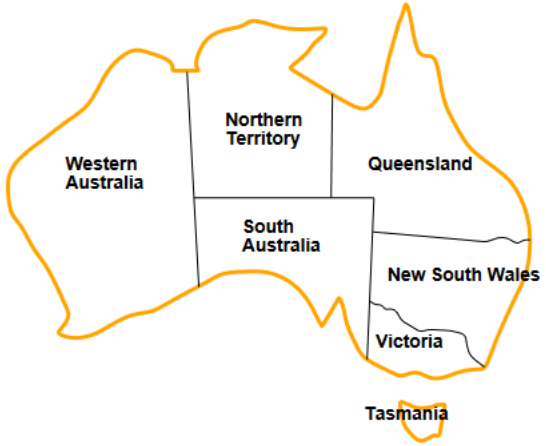


Forward checking

Idea: Keep track of remaining legal values for unassigned variables

Terminate search when any variable has no legal values

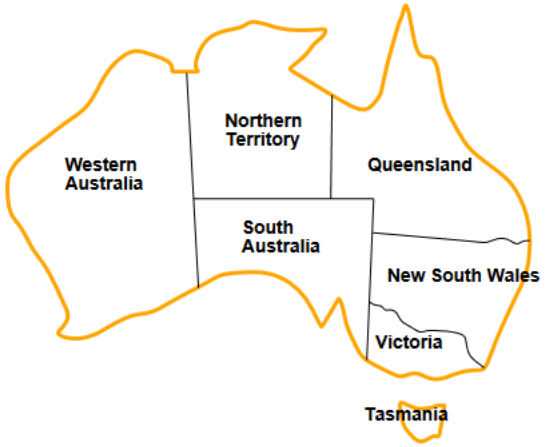




Forward checking

Idea: Keep track of remaining legal values for unassigned variables

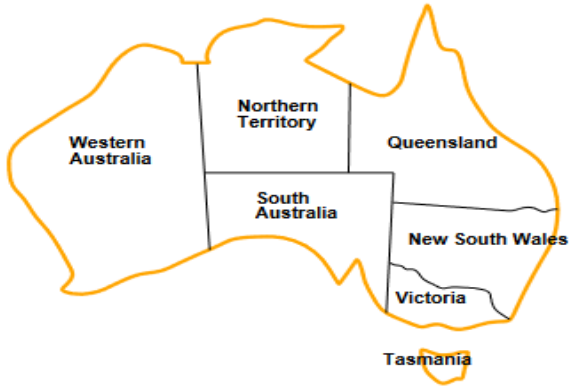
Terminate search when any variable has no legal values



Forward checking

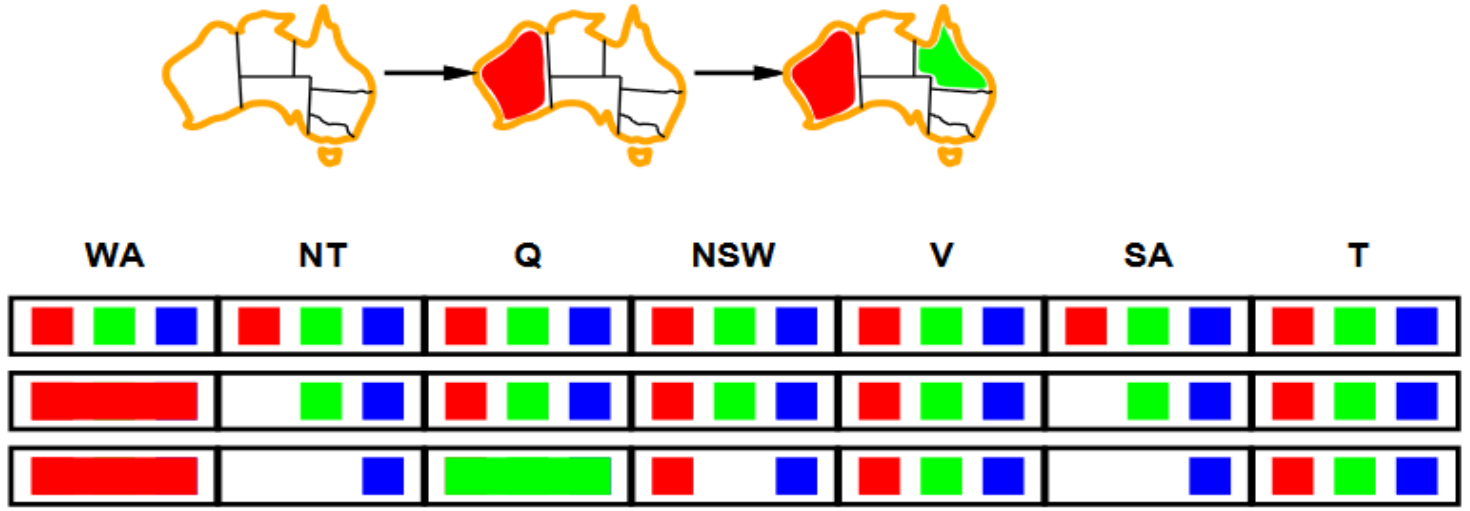
Idea: Keep track of remaining legal values for unassigned variables

Terminate search when any variable has no legal values



Constraint propagation

- Techniques like Constraint Propagation (CP) and Forward Checking (FC) are in effect eliminating parts of the search space
- Forward checking checking does not detect all failures.



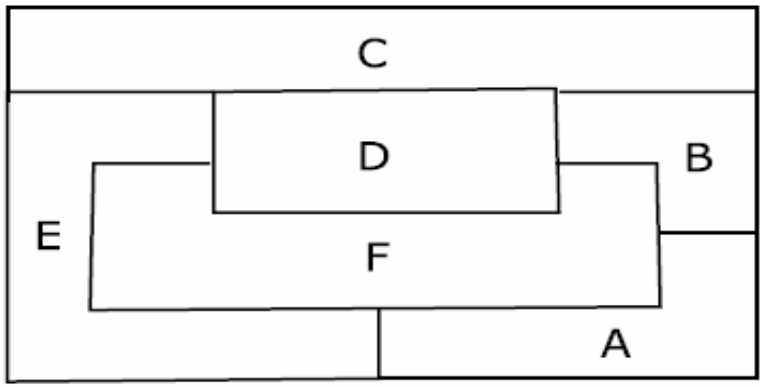
- NT and SA cannot both be blue!
- Constraint propagation goes further than FC by repeatedly enforcing constraints locally.

Constraint propagation

- When we assign a value to variable X, forward checking only checks variables that share a constraint with X, i.e. are adjacent in the constraint graph.
- Arc-consistency (AC) is a systematic procedure for constraining propagation
- A state is arc-consistent, if every variable has some value that is consistent with each of its constraints

Arc-consistency (AC)

Task: 3-color



Solution:

	A	B	C	D	E	F
	RGB	RGB	RGB	RGB	RGB	RGB
A=R	(R)	GB	RBG	RBG	GB	GB
B=G	(R)	(G)	~~R~~ B	R ~~B~~	G ~~B~~	B

$$D \neq F : D = \{R, \cancel{B}\}$$
$$E \neq F : E = \{G, \cancel{B}\}$$
$$C \neq D : C = \{\cancel{R}, B\}$$

Constraint propagation (CSP)

- **Constraint propagation**: Using the constraints to reduce the number of legal values for a variable, which in turn can reduce the legal values for another variable, and so on
- **Local consistency**: If we treat each variable as a node in a graph and each binary constraint as an arc, then the process of enforcing local consistency in each part of the graph causes inconsistent values to be eliminated throughout the graph.
- **Arc consistency:** A variable in a CSP is **arc-consistent** if every value in its domain satisfies the variable's binary constraints.
- X_i is arc-consistent with respect to another variable X_j if for every value in the current domain D_i there is some value in the domain D_j that satisfies the binary constraint on the arc (X_i, X_j).

CSP

- Backtracking search, a form of depth-first search, is commonly used for solving CSPs.
- **Commutativity:** CSPs are all commutative. A problem is commutative if the order of application of any given set of actions has no effect on the outcome.
- **Backtracking search**: A depth-first search that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign.
- Backtracking algorithm repeatedly chooses an unassigned variable, and then tries all values in the domain of that variable in turn, trying to find a solution. If an inconsistency is detected, then BACKTRACK returns failure, causing the previous call to try another value.

CSP

- An assignment that does not violate any constraints is called a **consistent** or legal assignment;
- A **complete assignment** is one in which every variable is assigned;
- A **solution** to a CSP is a consistent, complete assignment;
- A **partial assignment** is one that assigns values to only some of the variables.
- **Local search algorithms** for CSPs use a complete-state formulation: the initial state assigns a value to every variable, and the search change the value of one variable at a time.

END