CSP

SESSION 10

Constraint Satisfaction Problems

- Each constraint C_i is a pair $\langle S_i, R_i \rangle$
  - S_i is called the scope of the constraint; $S_i \subseteq X$

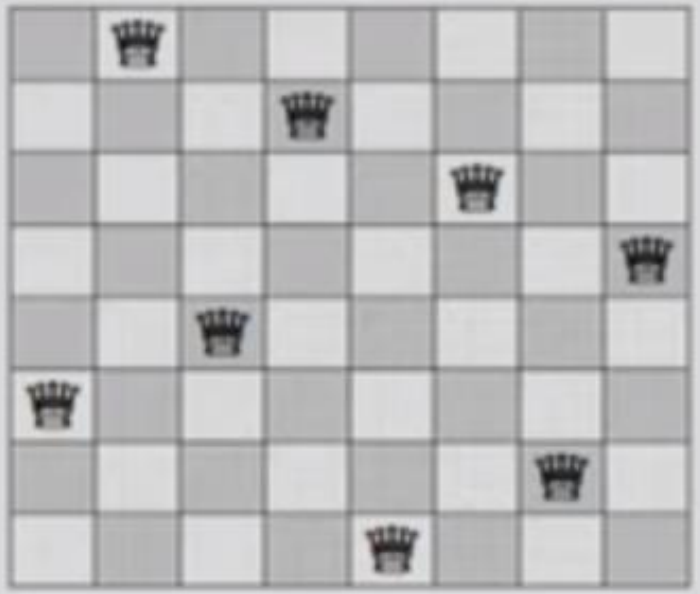
    For example: $S_i = \{X_{i1}, X_{i2}, \ldots\ldots X_{ip}\}$

    S_i involves some subset of the variables; R_i specifie the allowable combinations of values for that subset
  - $R_i \subseteq \{D_{i1} \times D_{i2} \times \ldots\ldots D_{ip}\}$
- A state of the problem is defined by an assignment of values to some or all of the variables, $\{X_i = v_i; X_j = v_j ; \ldots\ldots\}$.
- A solution to a CSP is a complete assignment that satisfies all the constraints.
  - An assignment that does not violate any constraints is called consistent or legal.
  - A complete assignment is one in which every variable is mentioned.

Constraint Satisfaction Problems

N-Queens Problem

Problem of placing *n* chess-queens on an *n*×*n* chessboard so that no two queens threaten each other.



For example: 4-Queens Problem

	X_1	X_2	X_3	X_4
1		✓		
2		✓		
3		×		
4		×		

$R_{12} = \{(1,3), (1,4), (2,4), (3,1), (4,2), (4, 1)\}$

R_{13} ; R_{14} ; R_{23} ; R_{24} ; R_{34}

Constraint Satisfaction Problems

- An assignment $\bar{a}$ is consistent if it satisfies constraints in its scope.

	X_1	X_2	X_3	X_4
1	Q			
2			Q	
3				
4		Q		

$\bar{a} = (1, 4, 2)$; is consistent. ✓

- An assignment that does not violate any constraints is called a consistent or legal assignment.
- Consistent solutions may not lead to a solution; refer to it as partial solution.

- Partial assignment mean that it is an assignment only to a subset of the variables.

Constraint Satisfaction Problems

Map-Colouring

Colouring each region of a map in such a way that no neighboring regions have the same color.

Variables X_i : WA, NT, Q, NSW, V , SA, and T.

Domain D_i : {red; green; blue}

Constraint C_i : Adjacent regions must have different colours.

R_{12}= {(red; green); (red; blue); (green; red); (green; blue); (blue; red); (blue; green)}

Varieties of CSP

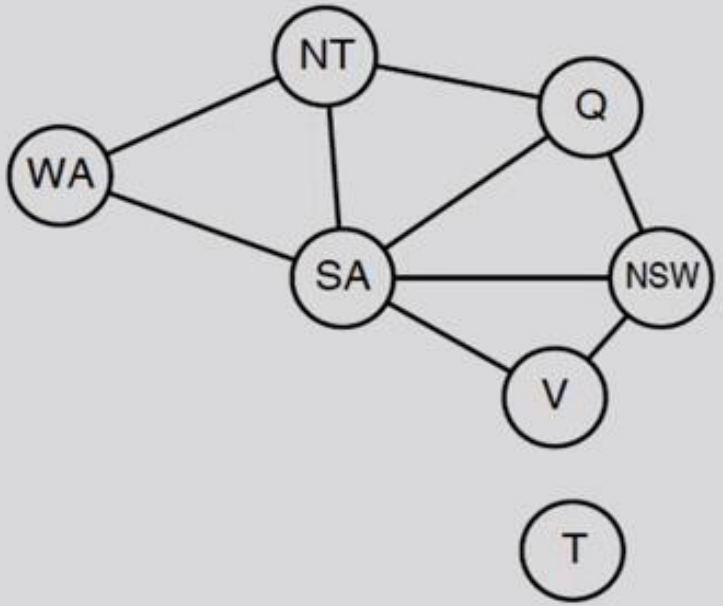
- Discrete variables
  - Finite domains (size d)
    - Boolean CSPs, including Boolean SAT (NP-complete)
    - $O(d^n)$ complete assignments
  - Infinite domains (integers, strings, etc.)
    - E.g., job scheduling, variables are start/end days for each job
    - Need a constraint language, e.g., startJob1 + 5 ≤ startJob3
    - linear constraints solvable, nonlinear undecidable
- Continuous variables
  - E.g., start/end times for Hubble Telescope observations
  - linear constraints solvable in polynomial time by linear programming methods

Varieties of Constraint

- **Unary Constraints** involve a single variable
  - e.g. C(X): X=2 C(Y): Y>5
- **Binary Constraints** involve pairs of variables
  - e.g. C(X,Y): X+Y<6
  - Can be represented by Constraint Graph
- **Higher-order constraints**: over 3 or more variables
  - We can convert any constraint into a set of binary constraints (may need some auxiliary variables).
- **Preferences / soft constraints**
  - E.g., red is better than green; Often representable by a cost for each variable assignment.

Constraint Graph

- Binary CSP – each constraint relates at most two variables.
- Constraint Graph - nodes correspond to variables and arcs correspond to constraints.

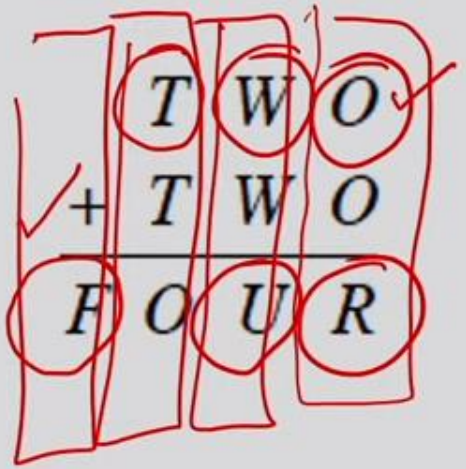


- General Purpose CSPs use the graph structure to speed-up search

Higher-order Constraints

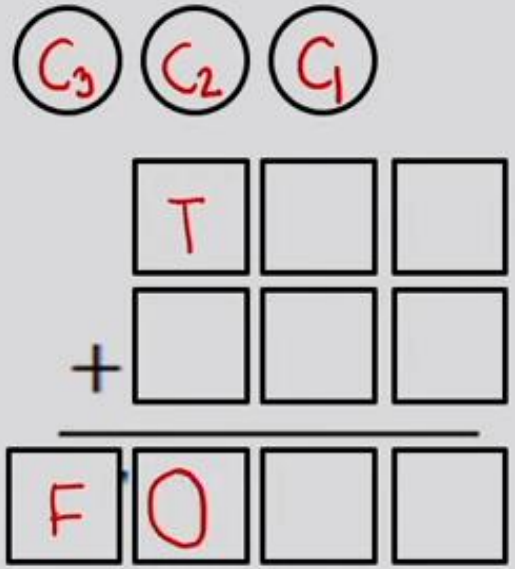
Cryptarithmetic Puzzle

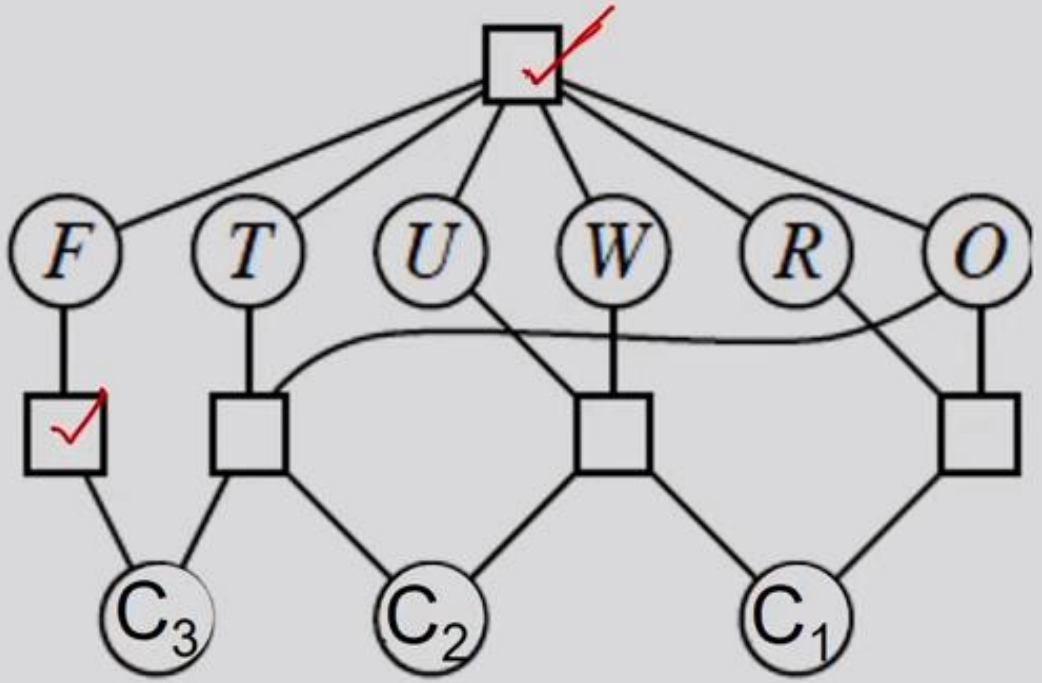
Each letter stands for a distinct digit; the aim is to find a substitution of digits for letters such that the resulting sum is arithmetically correct, with the added restriction that no leading zeroes are allowed.



- Six variable constraint for the above example.
- The addition constraints on the four columns of the puzzle also involve several variables
- Higher-order constraints can be represented in a constraint hypergraph.

Cryptarithmetic Puzzle





- The constraint hypergraph for the cryptarithmetic problem, shows the digit constraint as well as the column addition constraints.
- Each constraint is a square box connected to the variables it constrains.

Standard Search Formulation

- Let's start with a basic, naive approach and then improve it
- States are defined by the values assigned so far
  - Initial state: the empty assignment, { }
  - Successor function: assign a value to an unassigned variable that does not conflict with current assignment; fail if no legal assignments.
  - Goal test: the current assignment is complete

Note:

1. This is the same for all CSPs!
2. Every solution appears at depth n with n variables
3. Path is irrelevant, so can also use complete-state formulation
4. Domain of size d, branching factor $b = (n - \ell)\, d$ at depth ℓ; $n! d^n$ leaves!

Backtracking Search

- CSPs can be solved by a specialized version of depth first search.
- Key intuitions:
  - We can build up to a solution by searching through the space of partial assignments.
  - Order in which we assign the variables does not matter - eventually they all have to be assigned.
  - If during the process of building up a solution we falsify a constraint, we can immediately reject all possible ways of extending the current partial assignment.

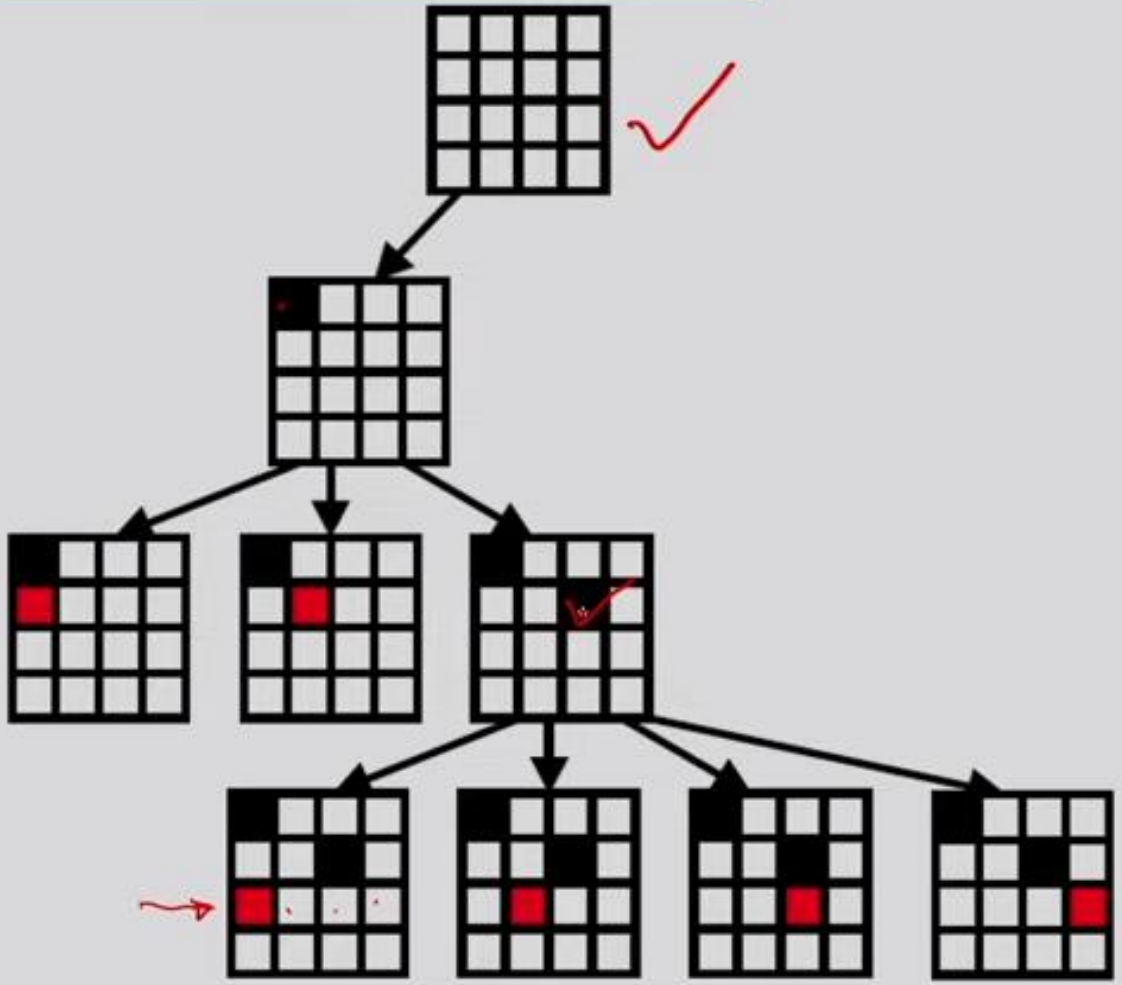
Backtracking Search

- Backtracking search is the basic uninformed algorithm for solving CSPs

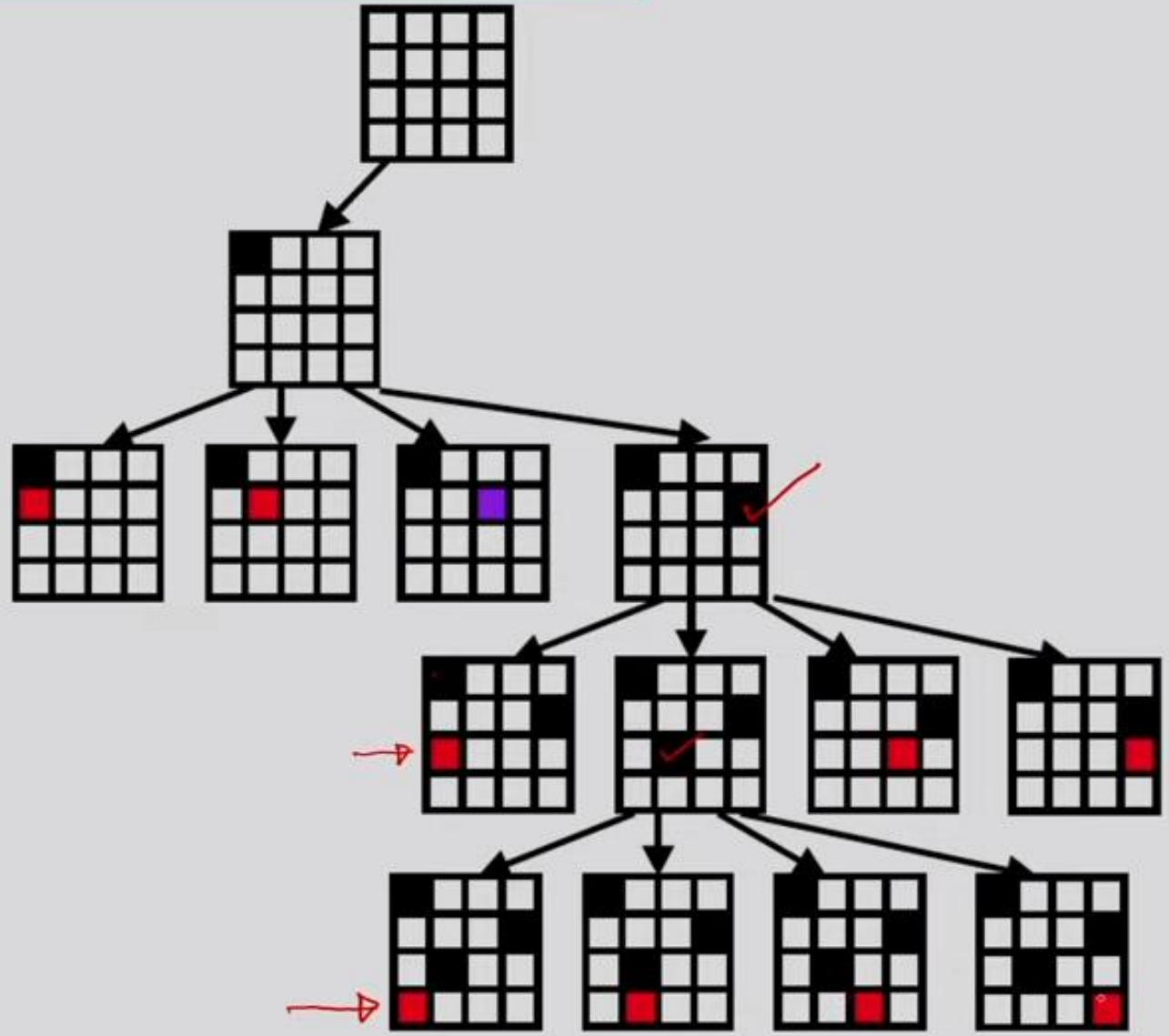
- Idea 1: One variable at a time
  - Variable assignments are commutative, so fix ordering
  - i.e., [WA = red then NT = green] same as [NT = green then WA = red]
  - Only need to consider assignments to a single variable at each step

- Idea 2: Check constraints as you go
  - i.e. consider only values which do not conflict with previous assignments
  - Might have to do some computation to check the constraints
  - "Incremental goal test"

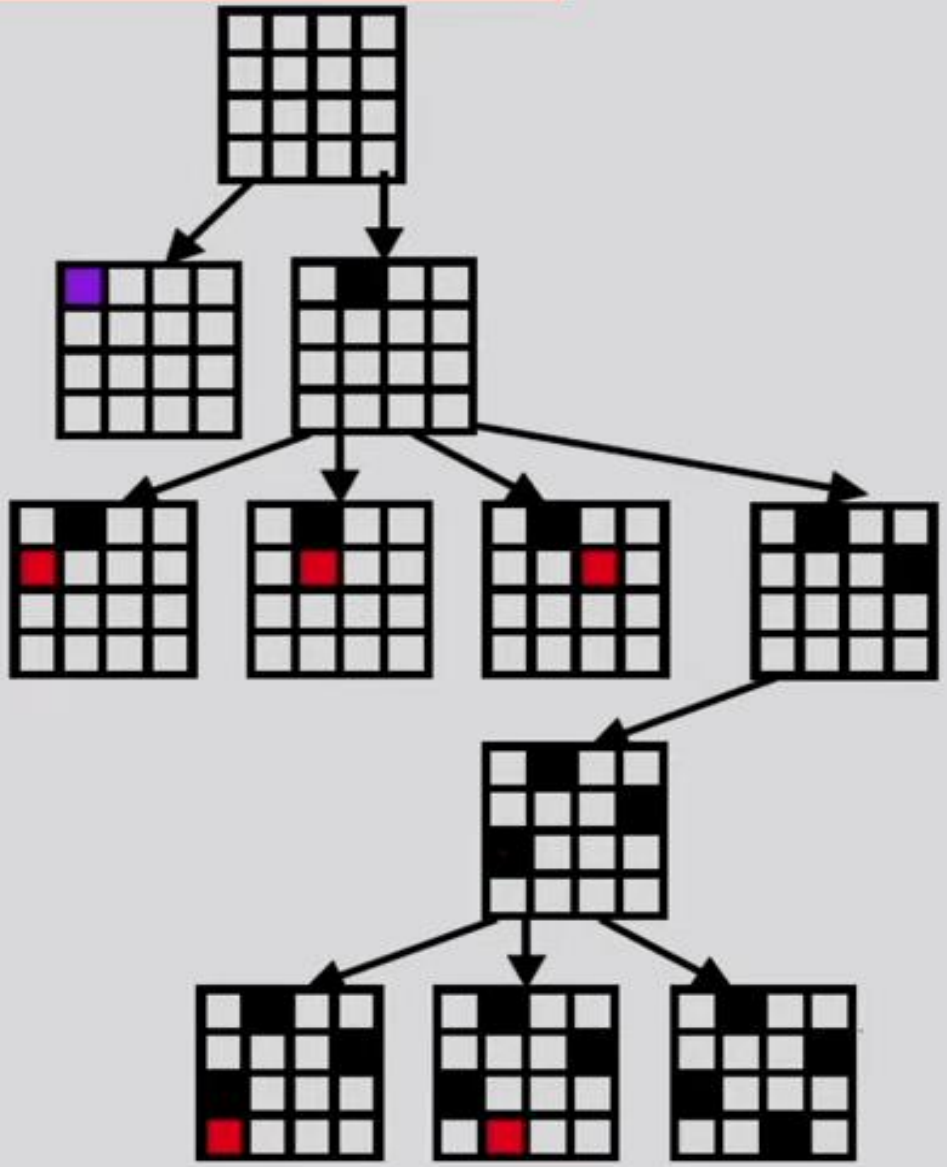
Backtracking Search



Backtracking Search



Backtracking Search



Backtracking Search

- Plain backtracking is an uninformed algorithm
  - Do not expect it to be very effective for large problems.
- We remedied the poor performance of uninformed search algorithms by supplying them with domain-specific heuristic functions derived from our knowledge of the problem.

Improving Backtracking Search

- We can solve CSPs efficiently without such domain-specific knowledge; general-purpose methods that address the following questions:
  1. Which variable should be assigned next, and in what order should its values be tried?
  2. What are the implications of the current variable assignments for the other unassigned variables?
  3. When a path fails i.e., a state is reached in which a variable has no legal values; can the search avoid repeating this failure in subsequent paths?

Variable and Value Ordering

- Minimum remaining values (MRV):
  - choose variable with the fewest legal values.

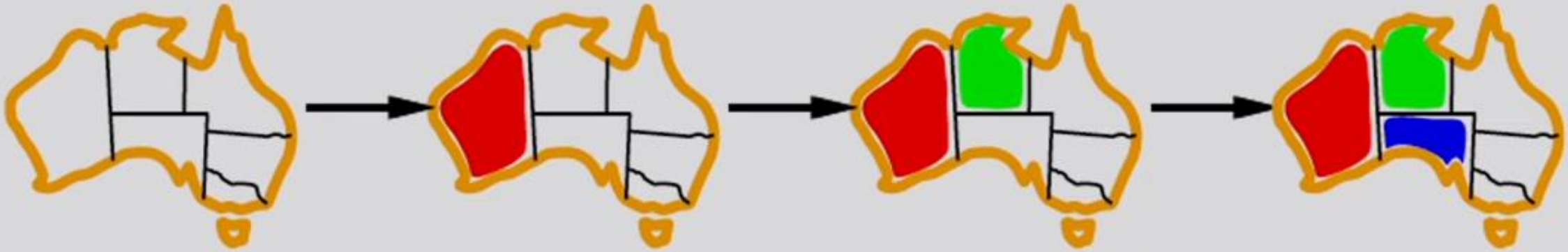
The backtracking algorithm contains the line

var SELECT-UNASSIGNED-VARIABLE(VARIABLES[*csp*], *assignment*, *csp*).

By default, SELECT-UNASSIGNED-VARIABLE simply selects the next unassigned variable in the order given by the list VARIABLES[csp]. This static variable ordering seldom results in the most efficient search.

Variable and Value Ordering

- Minimum remaining values (MRV):
  - choose variable with the fewest legal values.



After the assignments for WA=red and NT =green, there is only one possible value for SA, so it makes sense to assign SA=blue next rather than assigning Q.

It also has been called the "most constrained variable" or "fail-first" heuristic, the latter because it picks a variable that is most likely to cause a failure soon, thereby pruning the search tree.

Variable and Value Ordering

Degree Heuristics:

- choose the variable with the most constraints on remaining vars.

The MRV heuristic does not help at all in choosing the first region to colour in the above map, because initially every region has three legal colours.

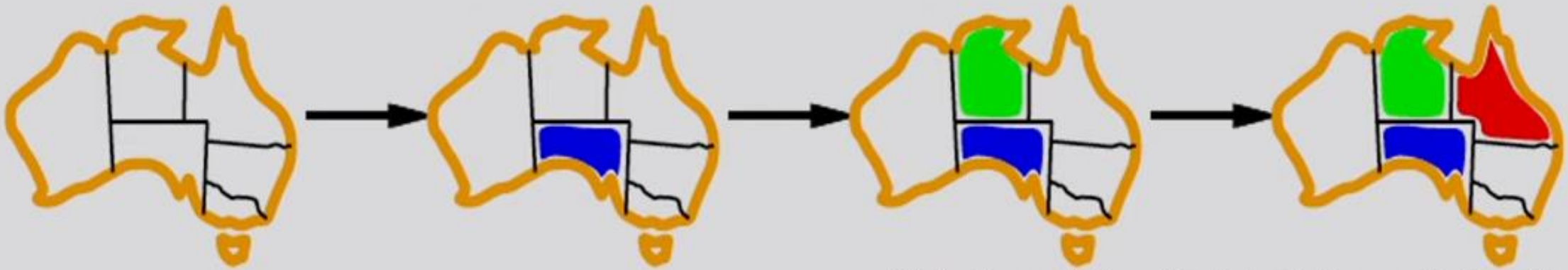
The degree heuristic comes in handy!

It attempts to reduce the branching factor on future choices by selecting the variable that is involved in the largest number of constraints on other unassigned variables.

Variable and Value Ordering

Degree Heuristics:

- choose the variable with the most constraints on remaining vars.



SA is the variable with highest degree, 5; the other variables have degree 2 or 3, except for T, which has 0.

Applying the degree heuristic solves the problem without any false steps —choose any consistent colour at each choice point and still arrive at a solution with no backtracking.

Variable and Value Ordering

Least Constraining value:

- choose the the one that rules out the fewest values in the remaining variables.

Once a variable has been selected, the algorithm must decide on the order in which to examine its values. It prefers the value that rules out the fewest choices for the neighboring variables in the constraint graph.

The least-constraining-value heuristic can be effective!

In general, the heuristic is trying to leave the maximum flexibility for subsequent variable assignments.

Variable and Value Ordering

Least Constraining value:

- choose the the one that rules out the fewest values in the remaining variables.

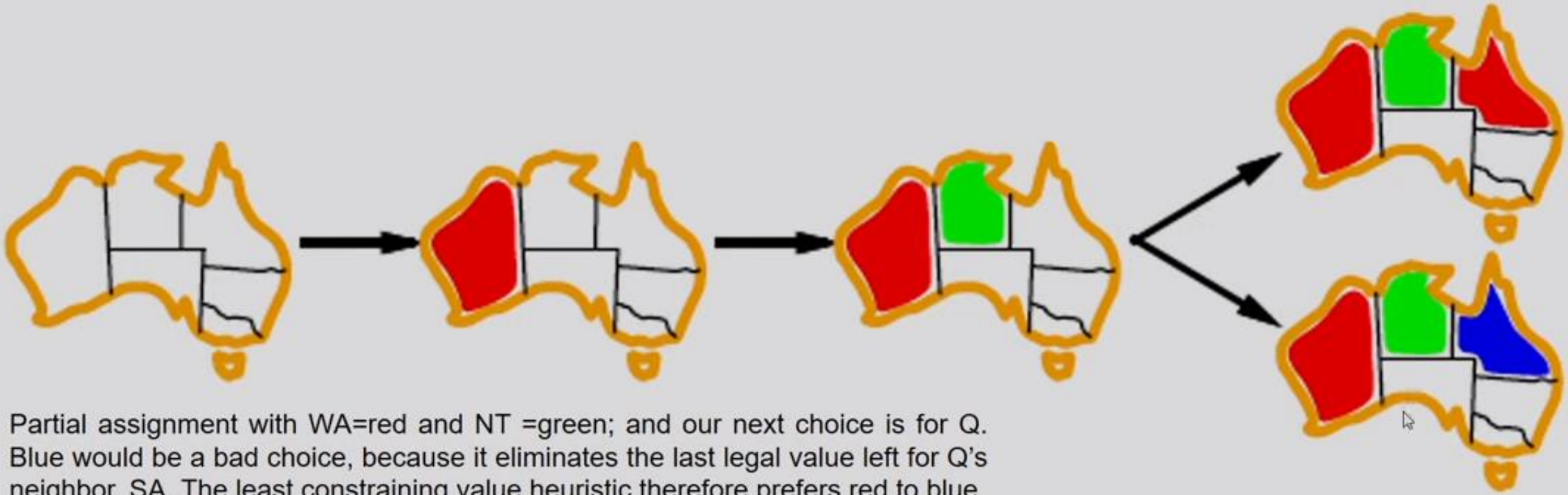
Once a variable has been selected, the algorithm must decide on the order in which to examine its values. It prefers the value that rules out the fewest choices for the neighboring variables in the constraint graph.

The least-constraining-value heuristic can be effective!

In general, the heuristic is trying to leave the maximum flexibility for subsequent variable assignments.

Variable and Value Ordering

- Least Constraining value:
  - choose the the one that rules out the fewest values in the remaining variables.



Partial assignment with WA=red and NT =green; and our next choice is for Q. Blue would be a bad choice, because it eliminates the last legal value left for Q's neighbor, SA. The least constraining value heuristic therefore prefers red to blue.

Propagating Constraints

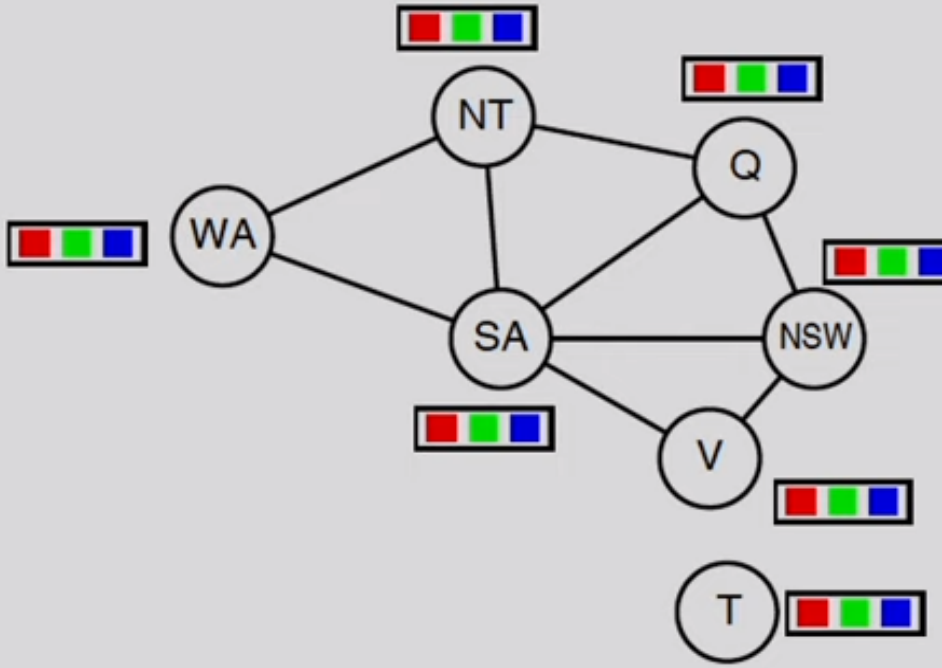
So far our search algorithm considered the constraints on a variable only at the time that the variable is chosen

var SELECT-UNASSIGNED-VARIABLE(VARIABLES[*csp*], *assignment*, *csp*).

Looking at some of the constraints earlier in the search, or even before the search has started, can drastically reduce the search space.

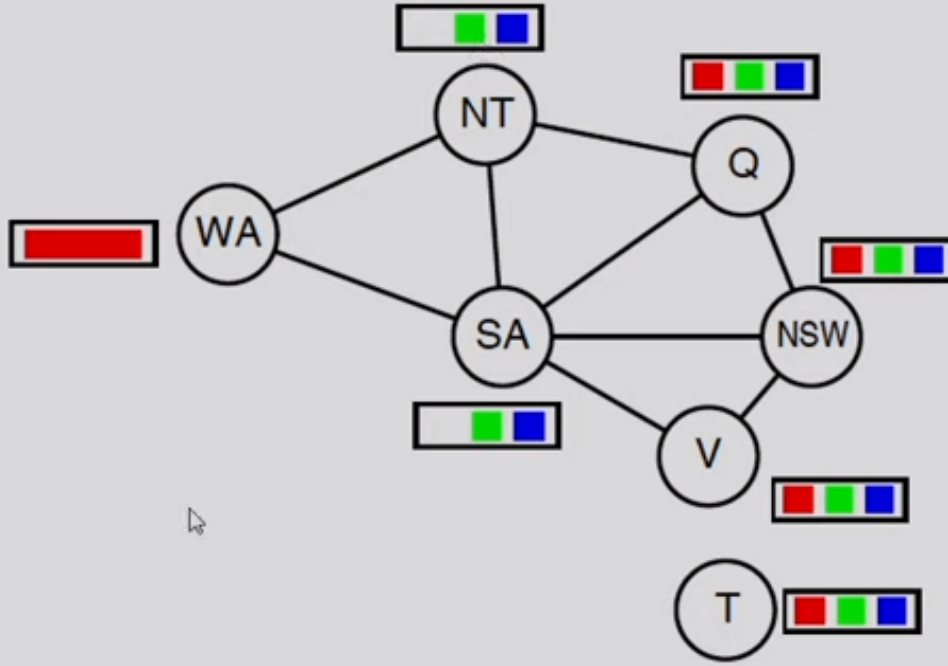
Forward Checking

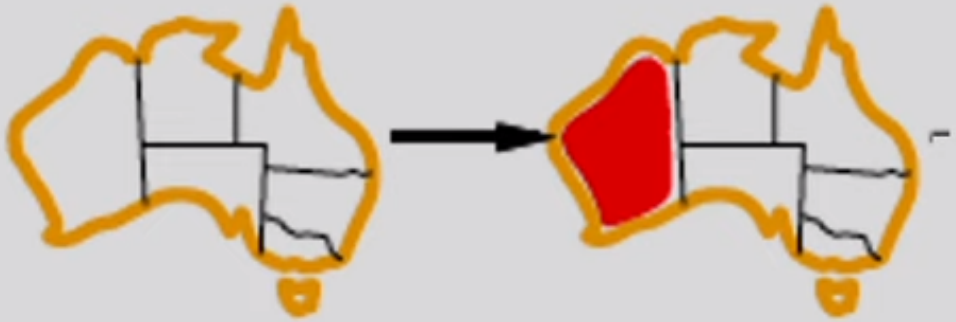
- Keep track of remaining legal values for unassigned variables. Terminate search when any variable has no legal values.



Forward Checking

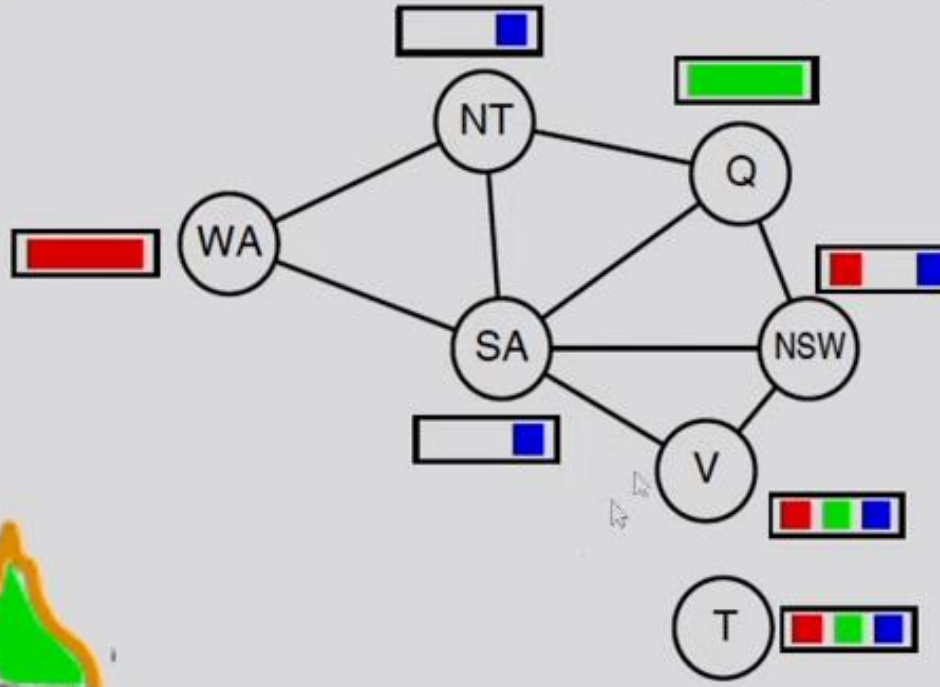
- Keep track of remaining legal values for unassigned variables. Terminate search when any variable has no legal values.





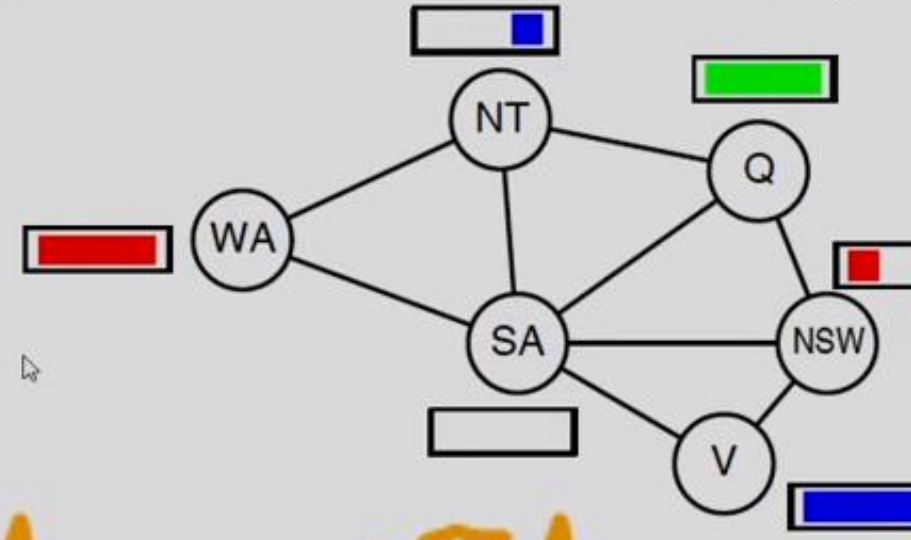
Forward Checking

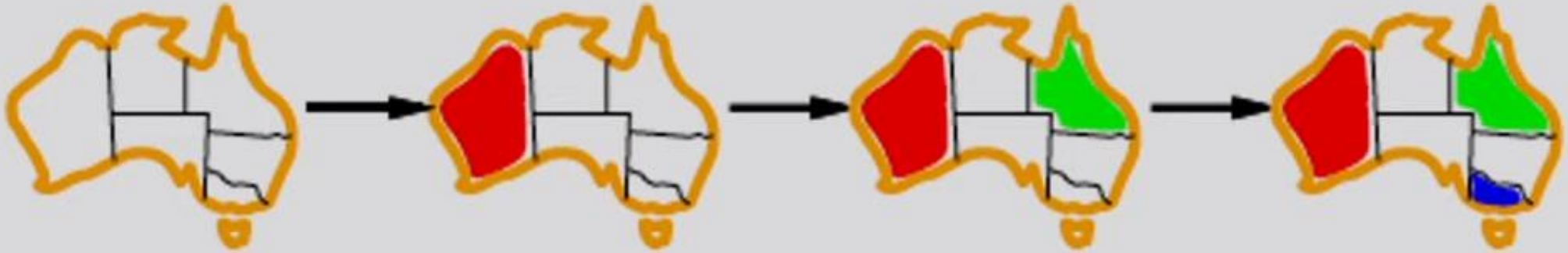
- Keep track of remaining legal values for unassigned variables. Terminate search when any variable has no legal values.



Forward Checking

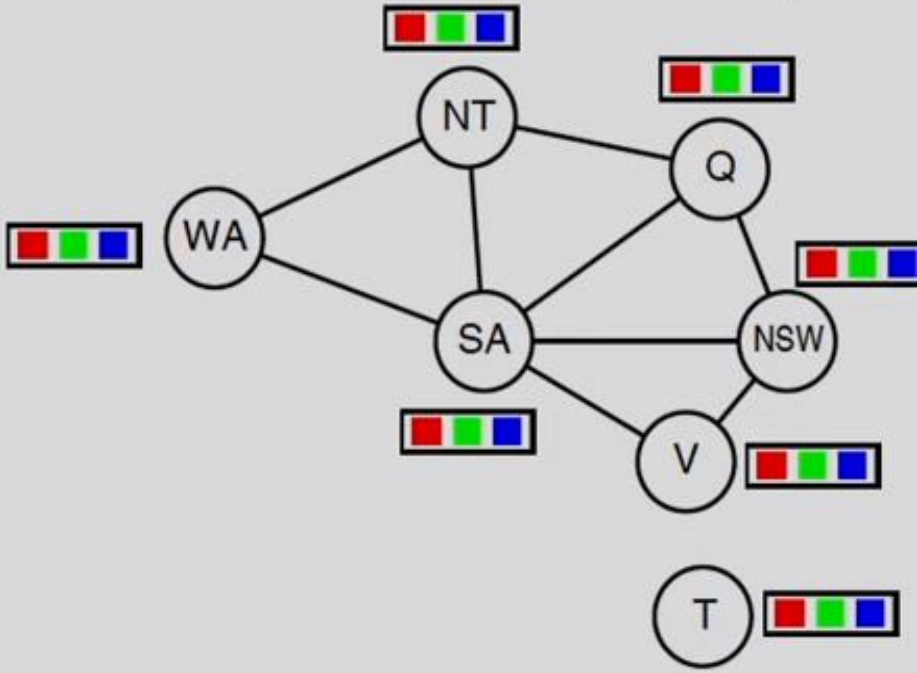
- Keep track of remaining legal values for unassigned variables. Terminate search when any variable has no legal values.





Constraint Propogation

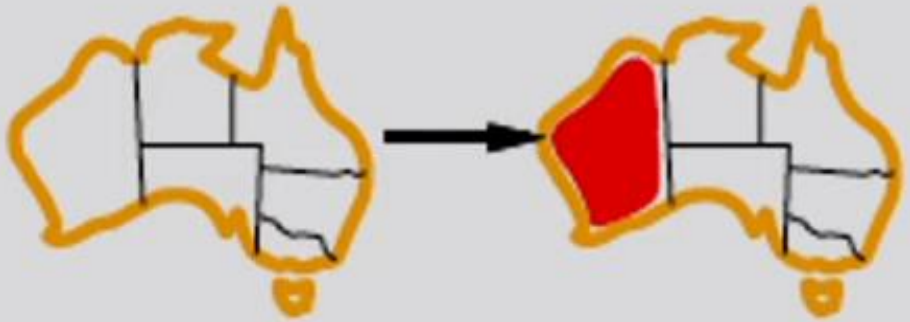
Propagate the implications of a constraint on one variable onto other variables to detect inconsistency.

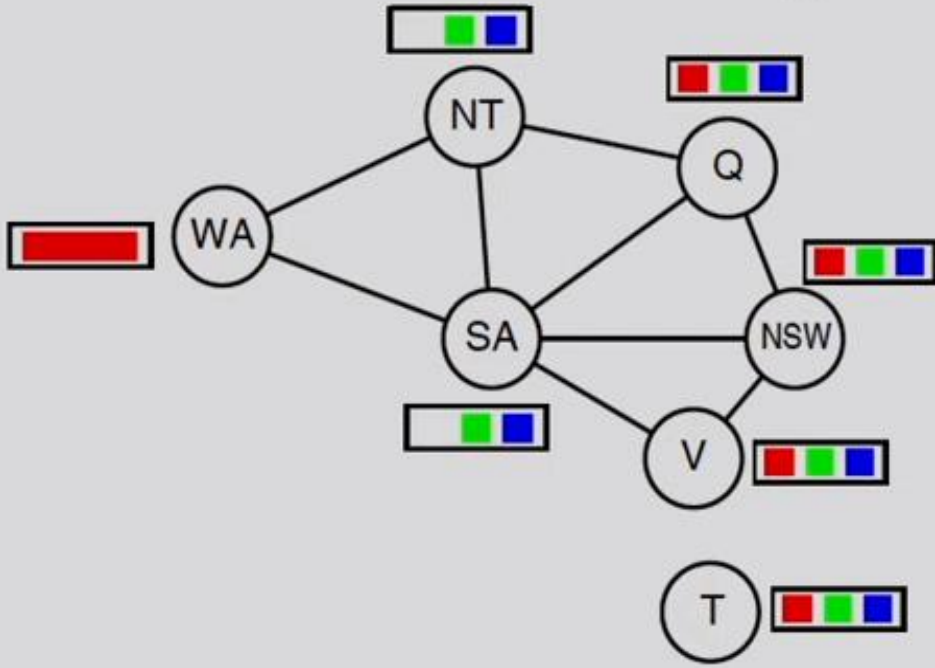


FC propagates information, but doesn't provide early detection for all failures:

Constraint Propogation

Propagate the implications of a constraint on one variable onto other variables to detect inconsistency.

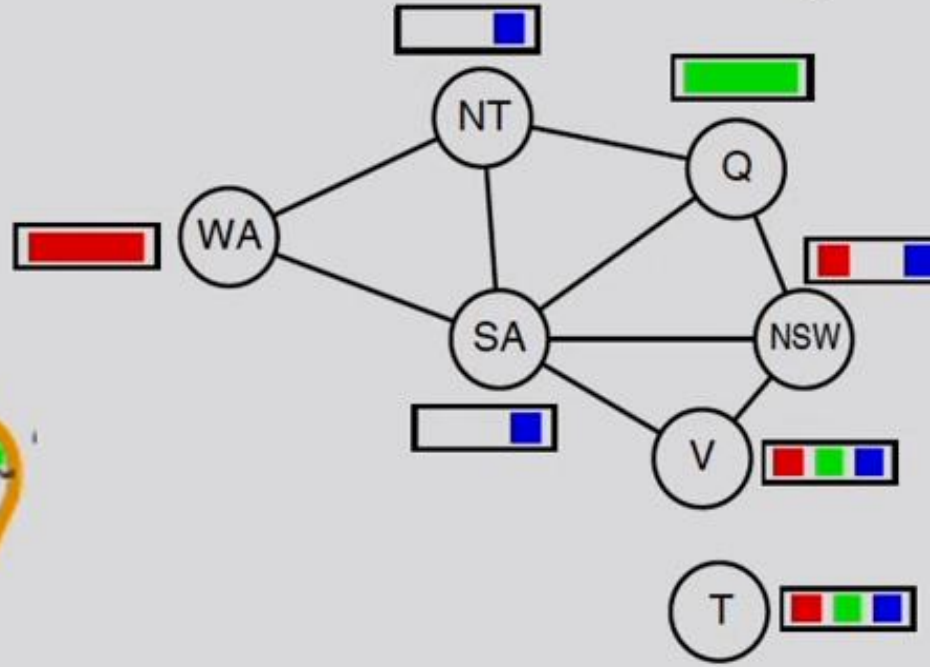




FC propagates information, but doesn't provide early detection for all failures:

Constraint Propogation

Propagate the implications of a constraint on one variable onto other variables to detect inconsistency.



FC propagates information, but doesn't provide early detection for all failures:

NT and SA cannot both be blue!

Constraint propagation repeatedly enforces constraints locally.

Constraint Propogation

- Constraint propagation is the general term for propagating the implications of a constraint on one variable onto other variables.
- And we want to do this fast: it is no good reducing the amount of search if we spend more time propagating constraints than we would have spent doing a simple search.
  - Arc Consistency - a fast method of constraint propagation that is substantially stronger than forward checking.