Izmir Institute of Technology

CENG 461 – Artificial Intelligence

Constraint Satisfaction Problems

2020-21 Fall Term

Introduction

- Previously we saw solving problems by searching in a space of states.
- Today, we will use a factored representation for each state: a set of **variables**, each of which has a value.
- A problem is solved when each variable has a value that satisfies all the **constraints** on the variable.
- A problem described in this way is called a Constraint Satisfaction Problem (CSP).

Defining CSPs

- A constraint satisfaction problem consists of three components, X, D, and C:
  - X is a set of variables, $\{X_1, \ldots, X_n\}$
  - D is a set of domains, $\{D_1, \ldots, D_n\}$ one for each variable
  - C is a set of constraints that specify allowable combinations of values.

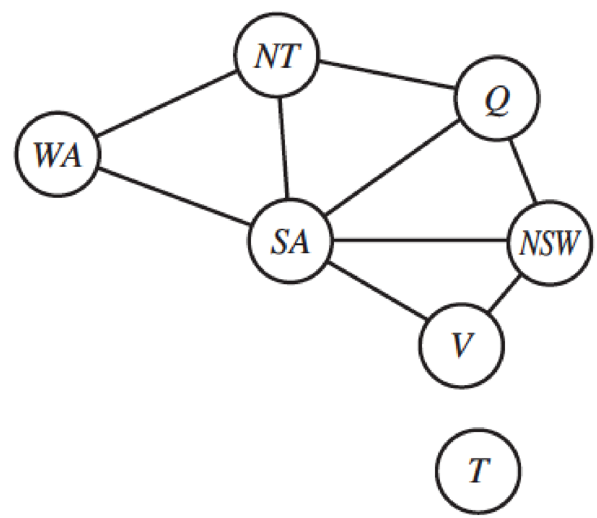
Example problem: Map coloring

- Coloring Australia map as a constraint satisfaction problem (CSP). The goal is to assign color to each region so that no neighbor e color.

Defining Map Coloring Problem

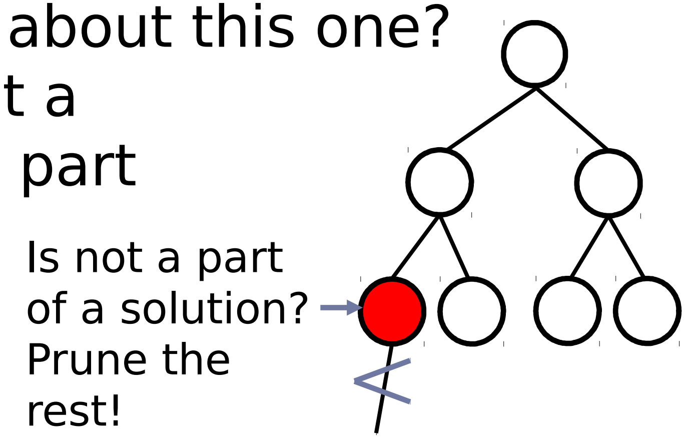
- X = {WA, NT, Q, NSW, V, SA, T}
- D_i = {red, green, blue}
- Since there are nine places where regions border, there are nine constraints:

  C={SA≠WA, SA≠NT, SA≠Q, SA≠NSW,
  SA≠V, WA≠NT, NT≠Q, Q≠NSW, NSW≠V}
- It is helpful to visualize CSP as a graph,
  where nodes correspond to variables and links correspond to constraints.



Why we define the problem as a CSP?

- In regular state-space search we can only ask: is this specific state a goal? No? What about this one? With CSPs, once we find out that a partial assignment can not be a part of a solution, we can discard further refinements of it.



- E.g. Once we have chosen SA={blue} we can conclude that none of the five neighboring variables can take the value blue. Regular state-space search procedure would consider 3^5=243 assignments for the five neighbors.
  With the techniques we will see, we never consider blue as a value, so we have only 2^5 =32 assignments to look at.

Another example: A cryptarithmetic puzzle

- Each letter in cryptarithmetic puzzle represents a different digit: Alldiff (F,T,U,W,R,O).
- Additional constraints

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

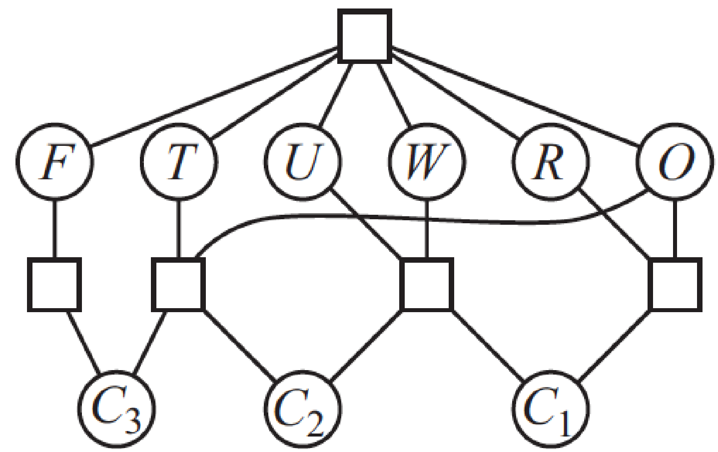
$$
\begin{aligned}
&O + O = R + 10 \cdot C_{10} \\
&C_{10} + W + W = U + 10 \cdot C_{100} \\
&C_{100} + T + T = O + 10 \cdot C_{1000} \\
&C_{1000} = F\ ,
\end{aligned}
$$

where C_{10}, C_{100}, and C_{1000} are auxiliary variables representing the digit carried over.

Another example: A cryptarithmetic puzzle

- These constraints can be represented in a constraint hypergraph.
- A hypergraph consists of ordinary nodes (the circles in the figure) and hypernodes (the squares), which represent **n**-ary constraints.
- Graph shows the Alldiff constraint (square box at the top) as well as the row addition constraints (four square boxes in the middle).

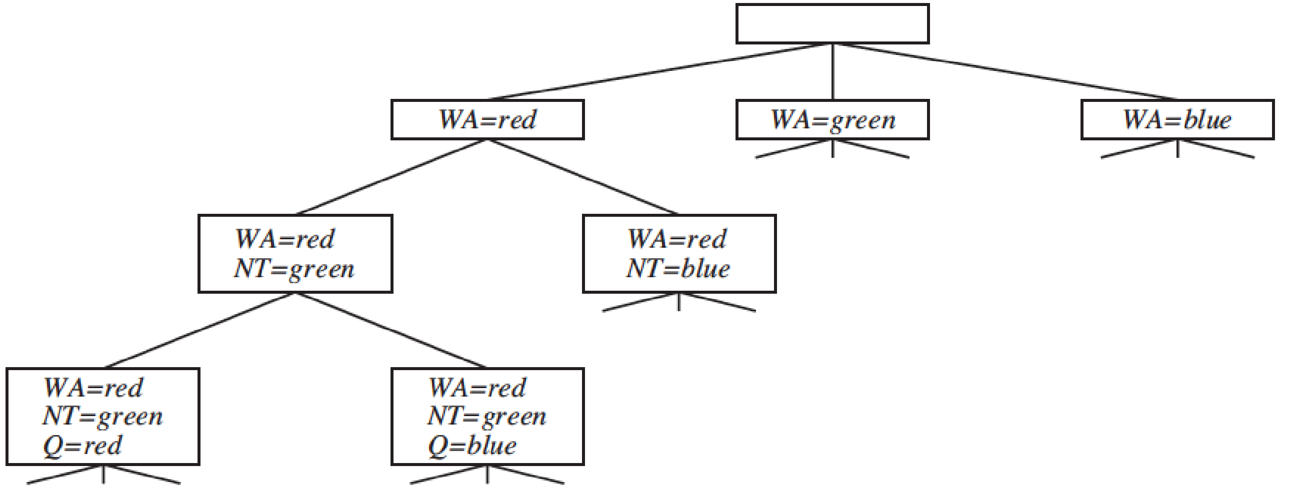
$$O + O = R + 10 \cdot C_{10}$$
$$C_{10} + W + W = U + 10 \cdot C_{100}$$
$$C_{100} + T + T = O + 10 \cdot C_{1000}$$
$$C_{1000} = F\ ,$$



Note: C_1 is C_{10}, C_2=C_{100}, C_3=C_{1000}

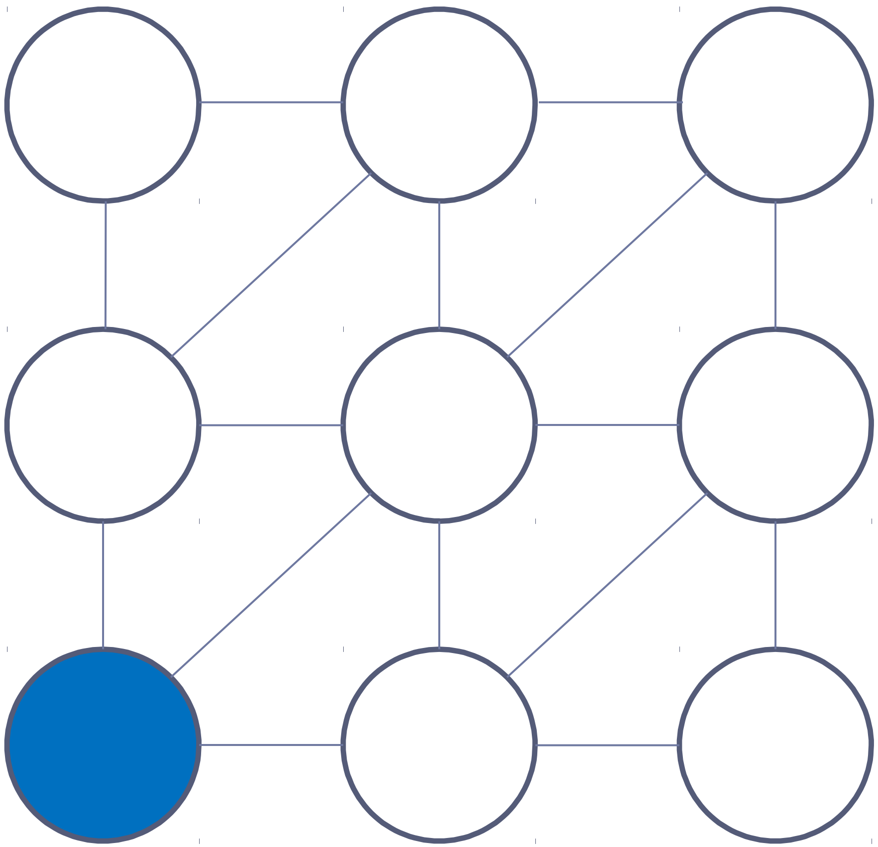
Backtracking Search for CSPs

- Backtracking search uses a depth-first search that
  - chooses values for one variable at a time
  - checks constraints as you go (backtracks when a variable has no legal values left to assign).
- Initial part of the search tree for the Australia problem is shown below where we assigned variables in the order WA,NT,Q, …



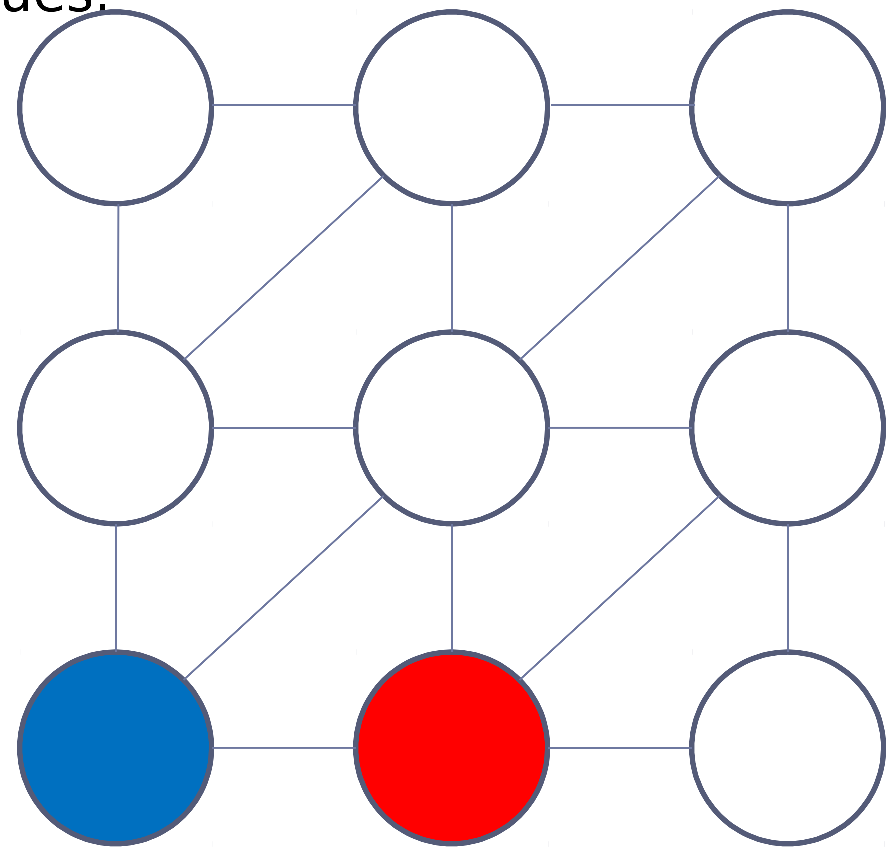
Backtracking Search for Coloring

- We have 9 variables, D_i = {red, green, blue},
- Constraint: Any two linked variables have different colors.



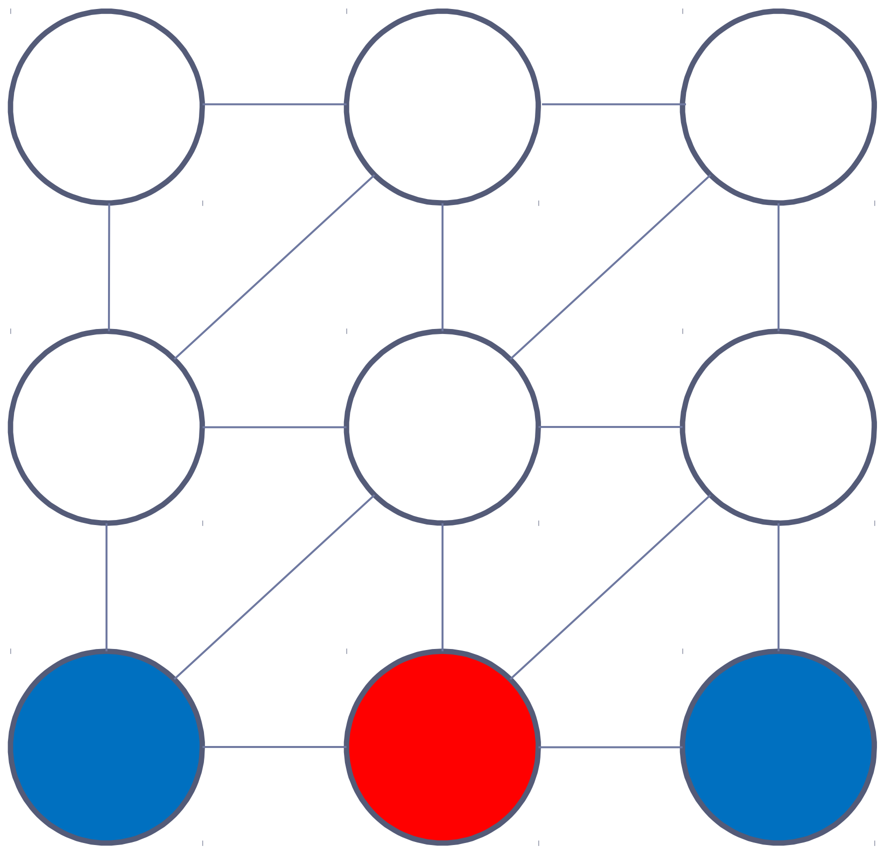
Backtracking Search for Coloring

- The second variable to color is the bottom-center one.
- Let’s say we color it Red, since it is one of the proper values.



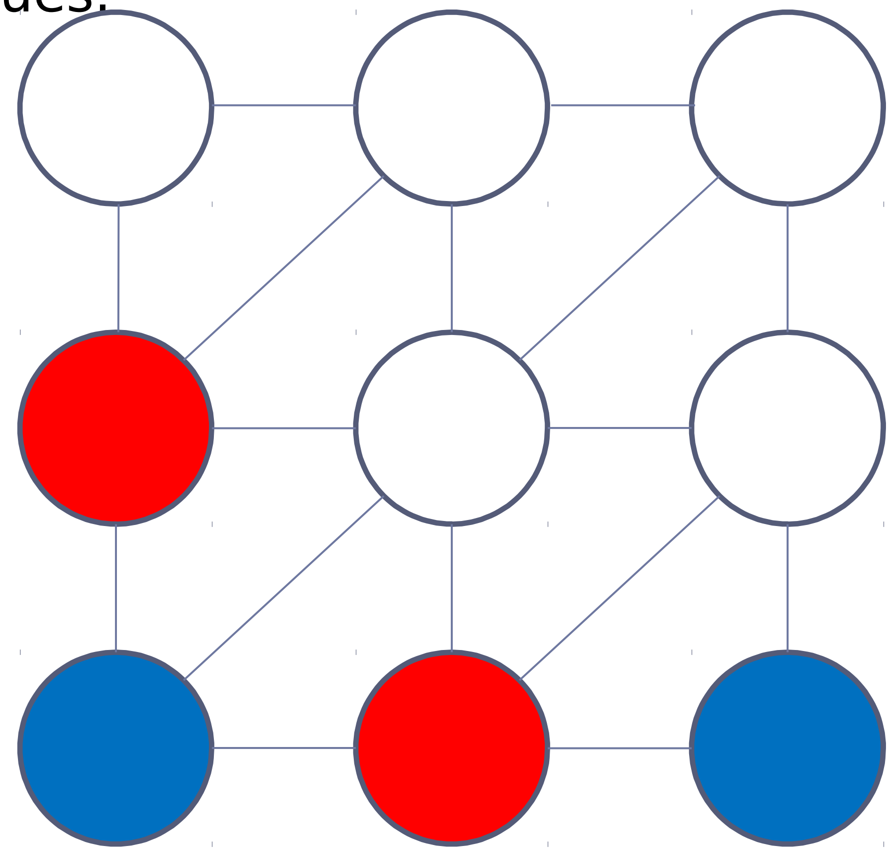
Backtracking Search for Coloring

- The third variable to color is the bottom-right one.
- Let’s say we color it Blue, since it is one of the proper values.



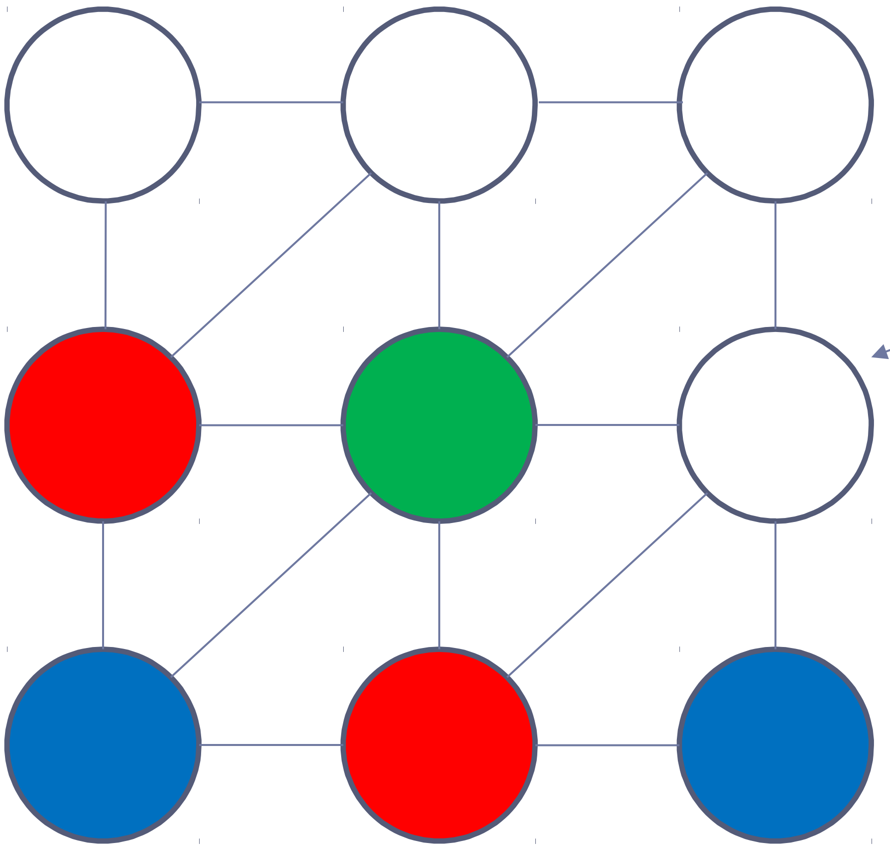
Backtracking Search for Coloring

- The fourth variable to color is the middle-left one.
- Let's say we color it Red, since it is one of the proper values.



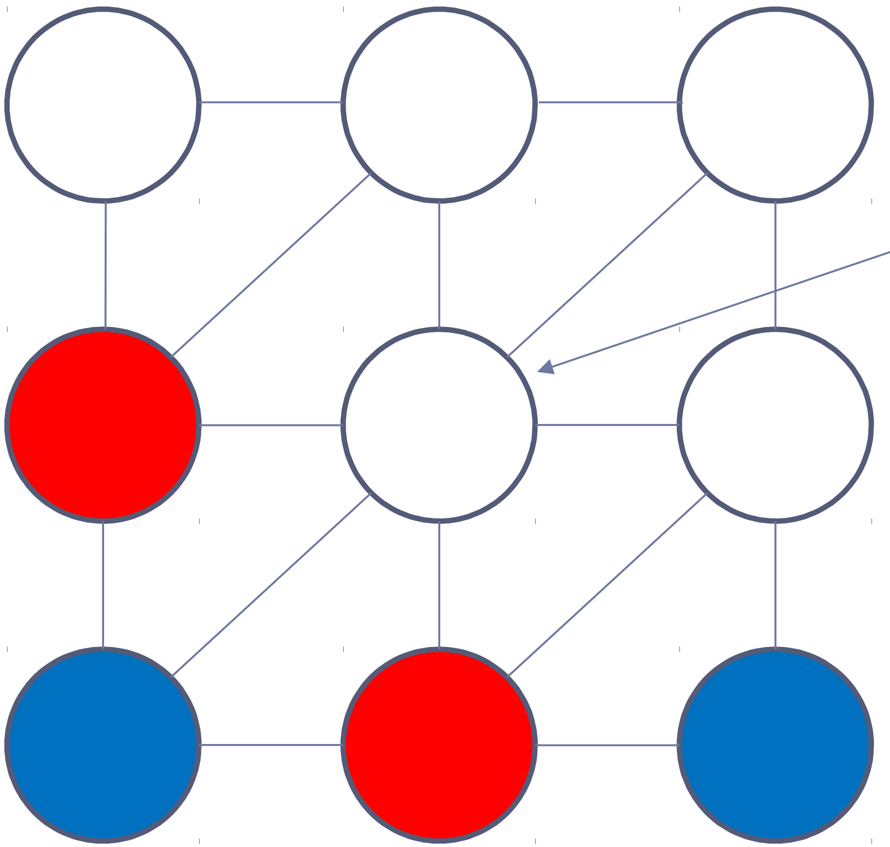
Backtracking Search for Coloring

- The fifth variable to color is the middle-center one.
- We have to color it in Green, since it is the only proper value.



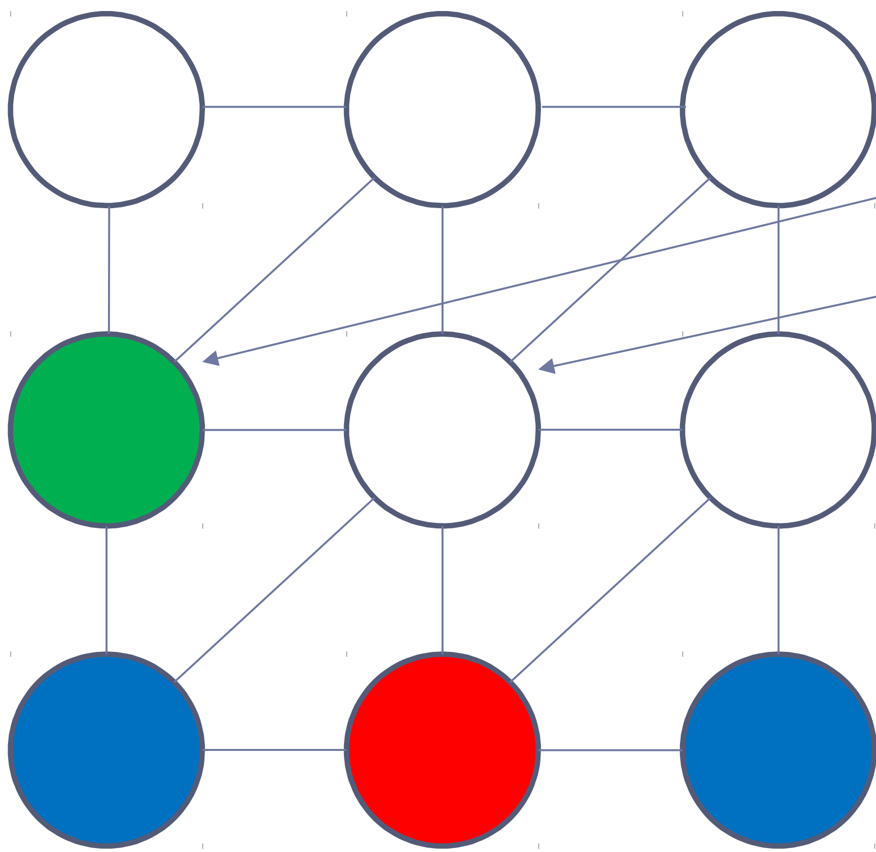
In the next step,
we realize that it is not going to work out because no proper value left for this box.
BACKTRACK!

Backtracking Search for Coloring



You are only able to give Green here (which didn't work).
So, BACKTRACK!

Backtracking Search for Coloring

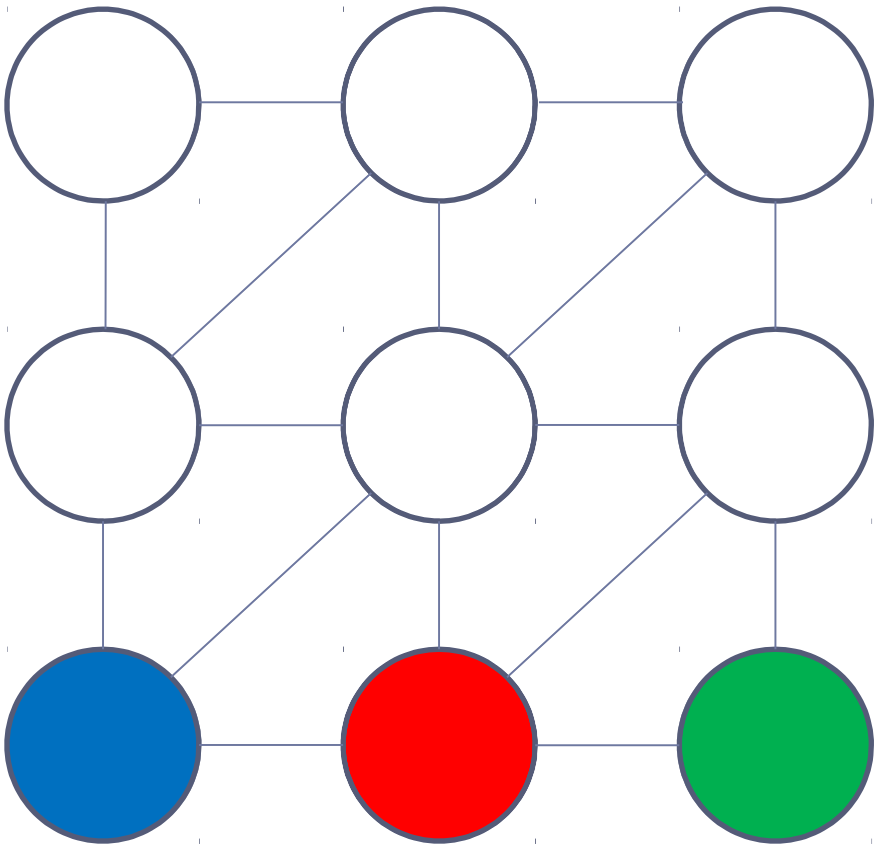


Assign Green here.

In the next step,
No proper value for here.
So,
BACKTRACK!

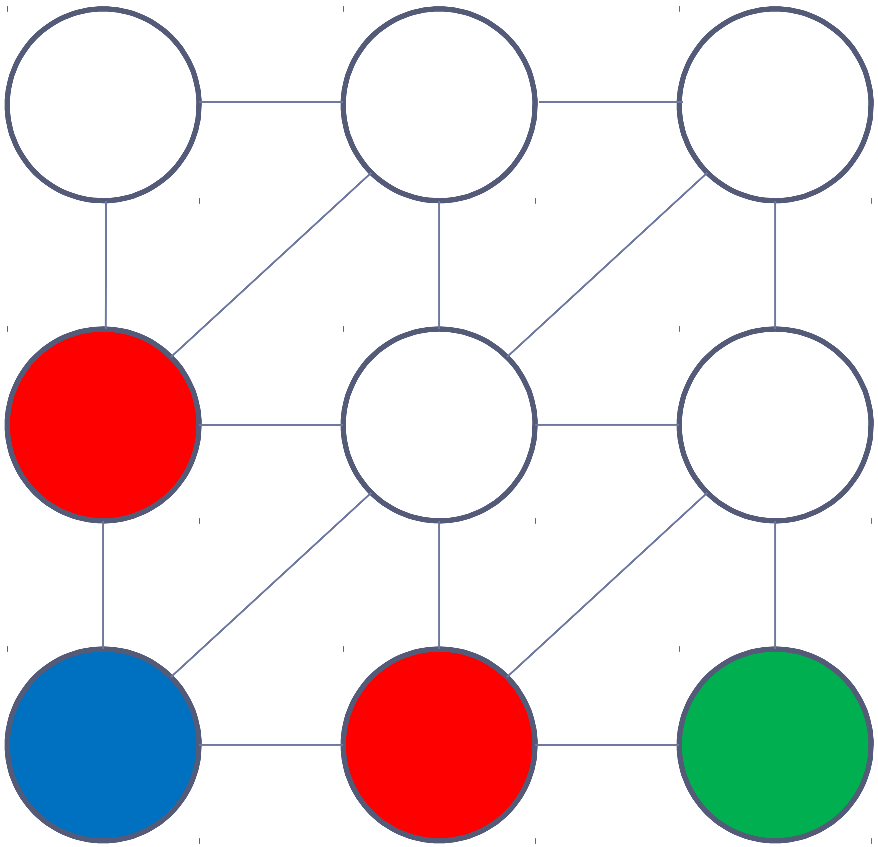
Backtracking Search for Coloring

- Change the value in the third variable.



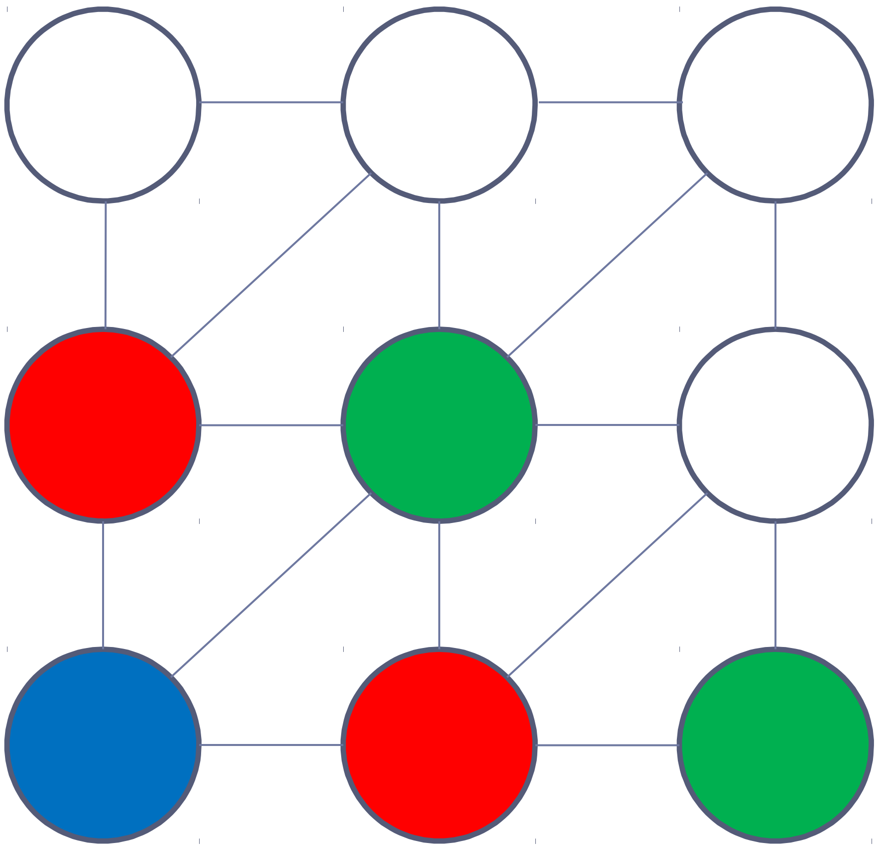
Backtracking Search for Coloring

- Now, we can try Red for the fourth variable again.
- It did not work before, but now the third variable is changed.



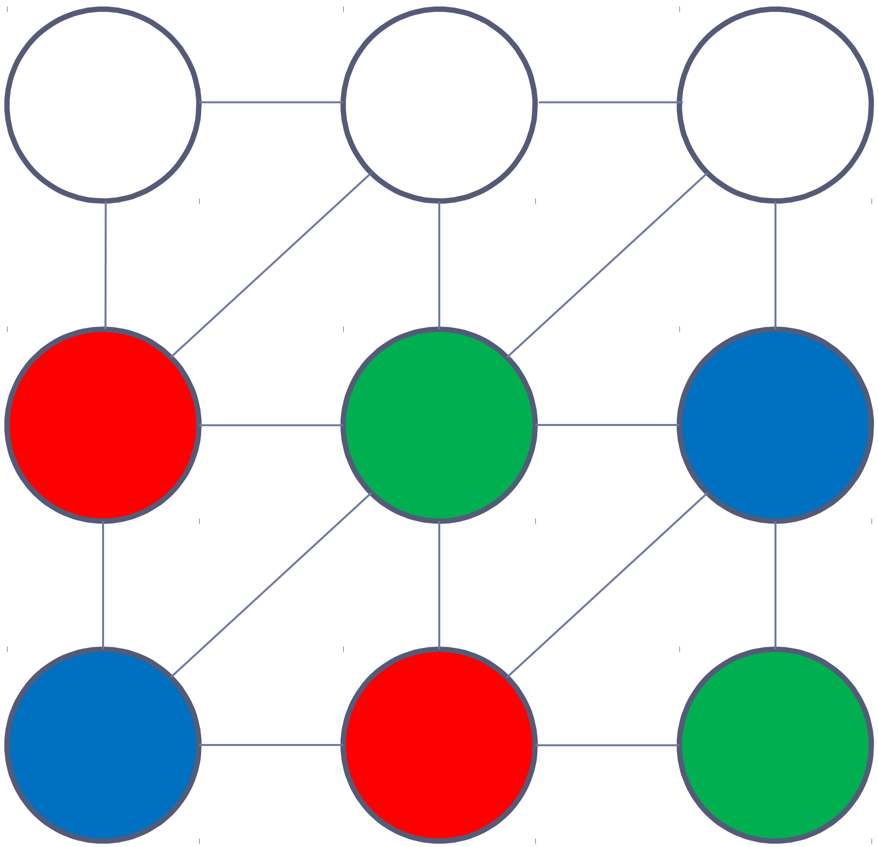
Backtracking Search for Coloring

- The only proper value for fifth variable is Green.



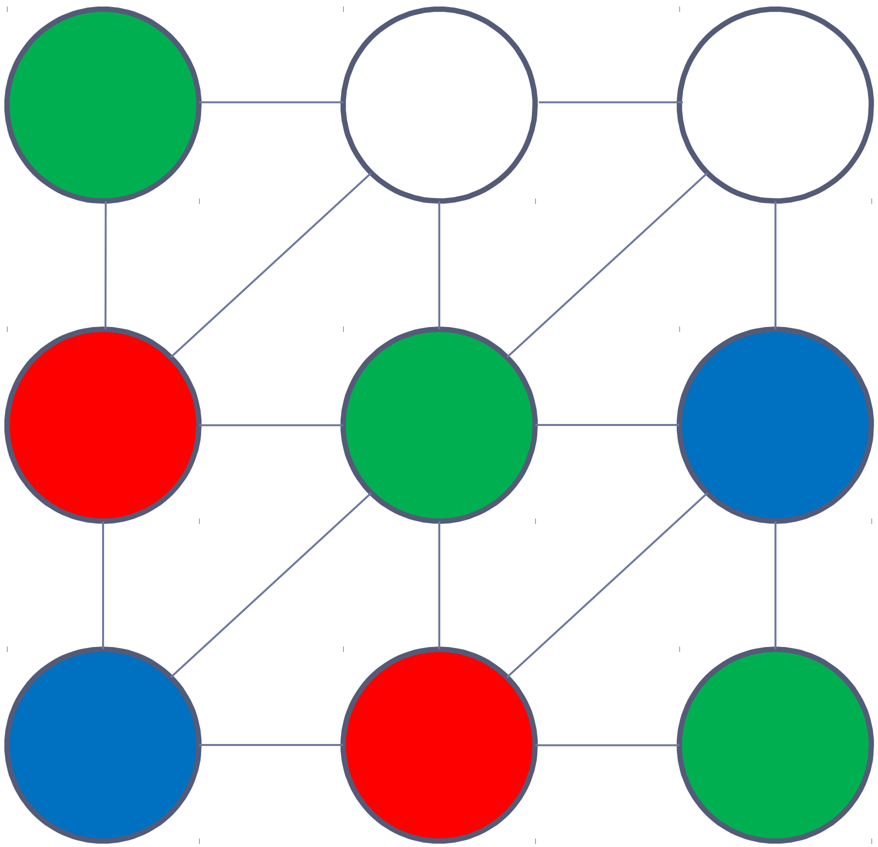
Backtracking Search for Coloring

- The only proper value for sixth variable is Blue.



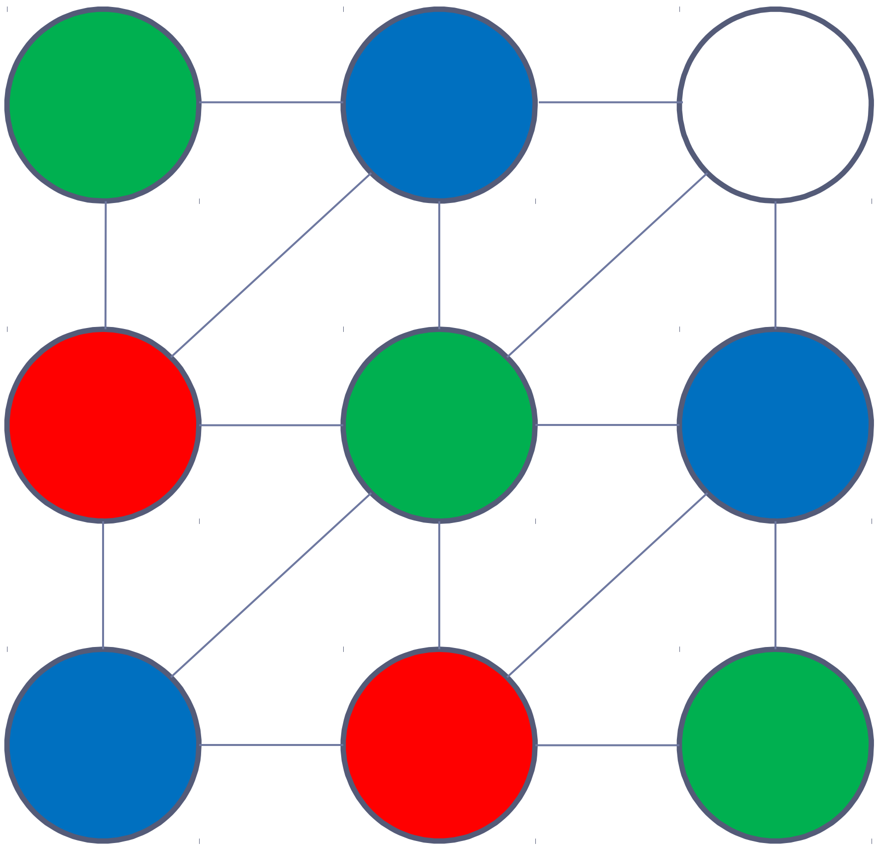
Backtracking Search for Coloring

- For the next one we can pick Green or Blue.
- Let’s pick Green, if it does not work, we will backtrack.



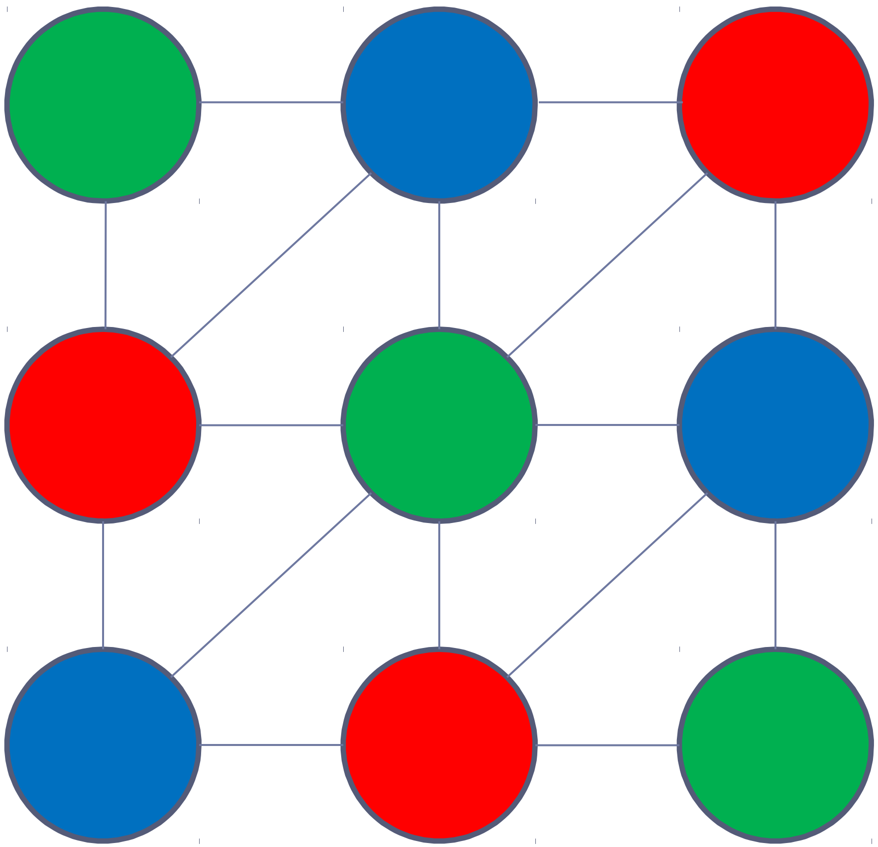
Backtracking Search for Coloring

- The only proper value for the next variable is Blue.



Backtracking Search for Coloring

- The only proper value for the next variable is Red.
- We reached a solution!



Improving Backtracking

- Can we detect failure early? Can we filter remaining nodes in the lower parts of the tree?
  - Filtering: Forward checking

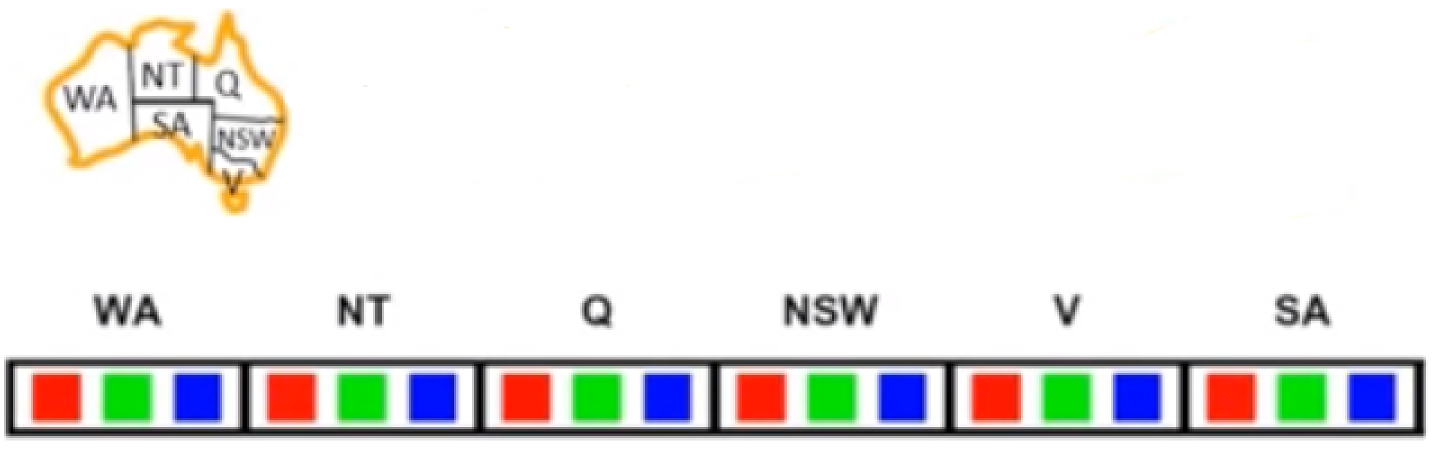
    When an assignment is made, for other variables cross-off (eliminate) values that violate a constraint.

  - Filtering: Arc consistency

    At each step, check consistency between all the variables.

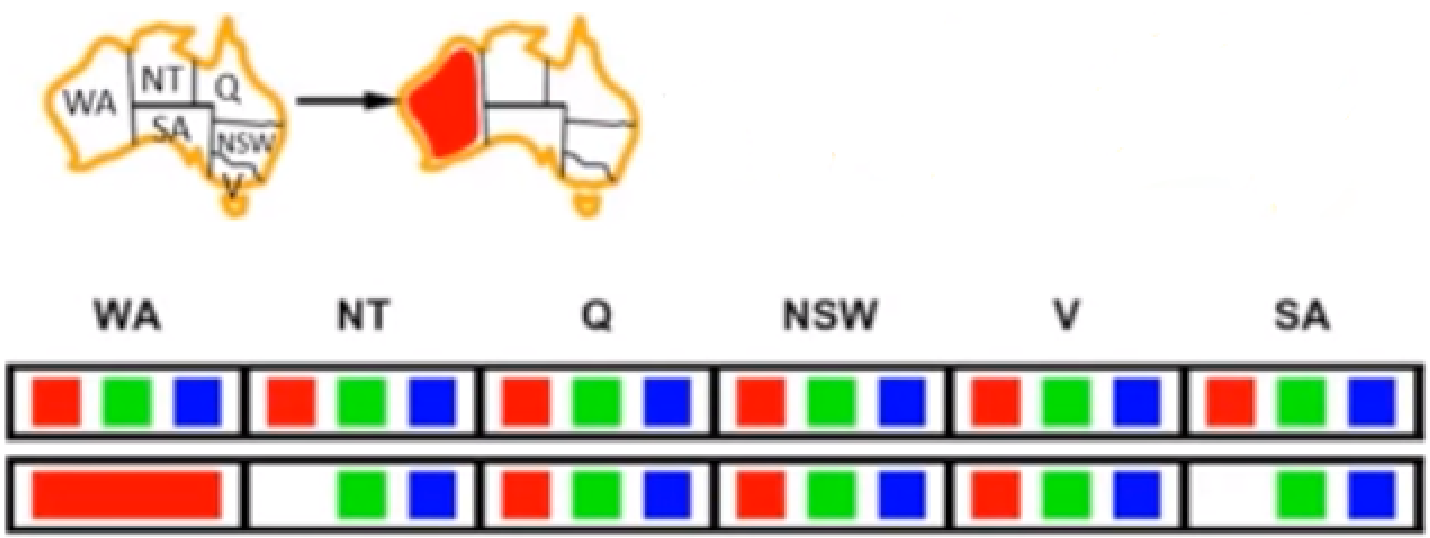
Filtering: Forward Checking

- Consider the Australia coloring problem.
- At each time step, all the remaining unassigned variables have their domains showed. When we make an assignment, we check remaining variables and cross-off violating values.
- Initial state looks like this:



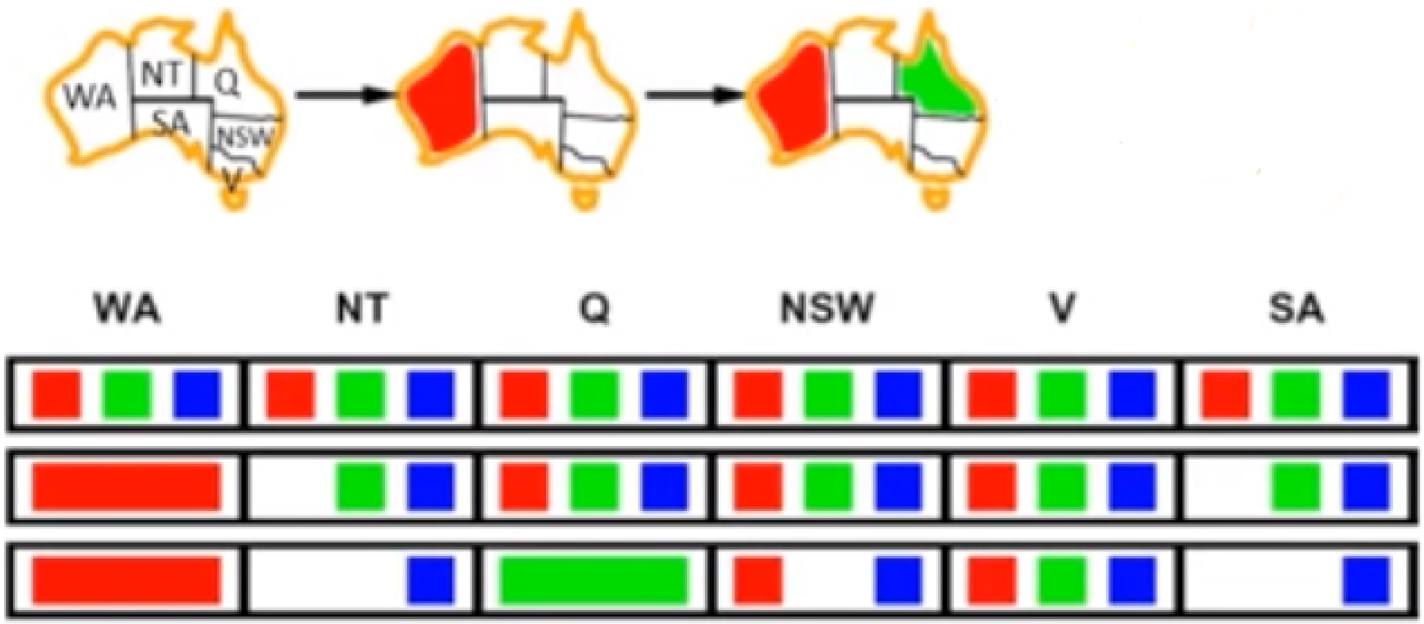
Filtering: Forward Checking

- Assume we assign WA in Red.
- All adjacent variables are checked, violating values are eliminated.
- NT and SA are not assigned, but their domains shrunk.



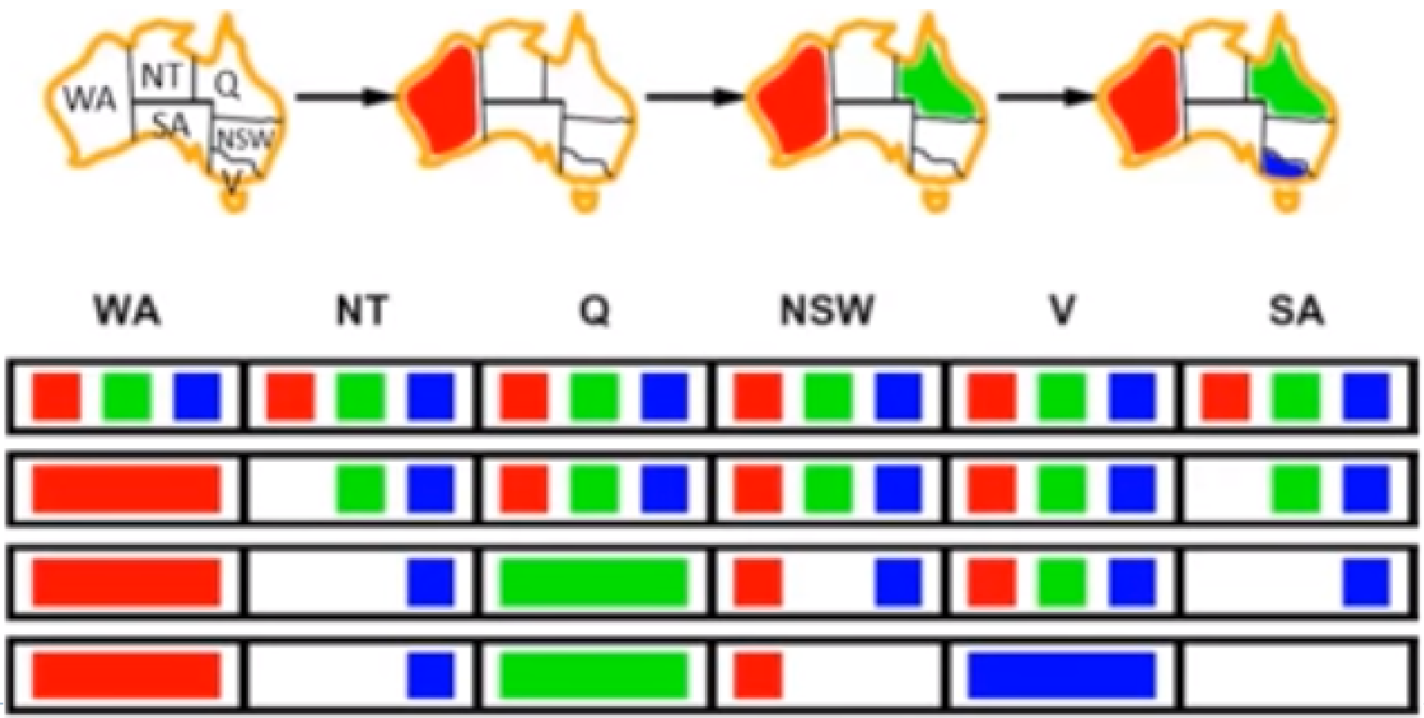
Filtering: Forward Checking

- Now, assume we color Q in Green (one of the proper values).
- Violating values are eliminated only from the adjacent variables. We don't look further at this moment.



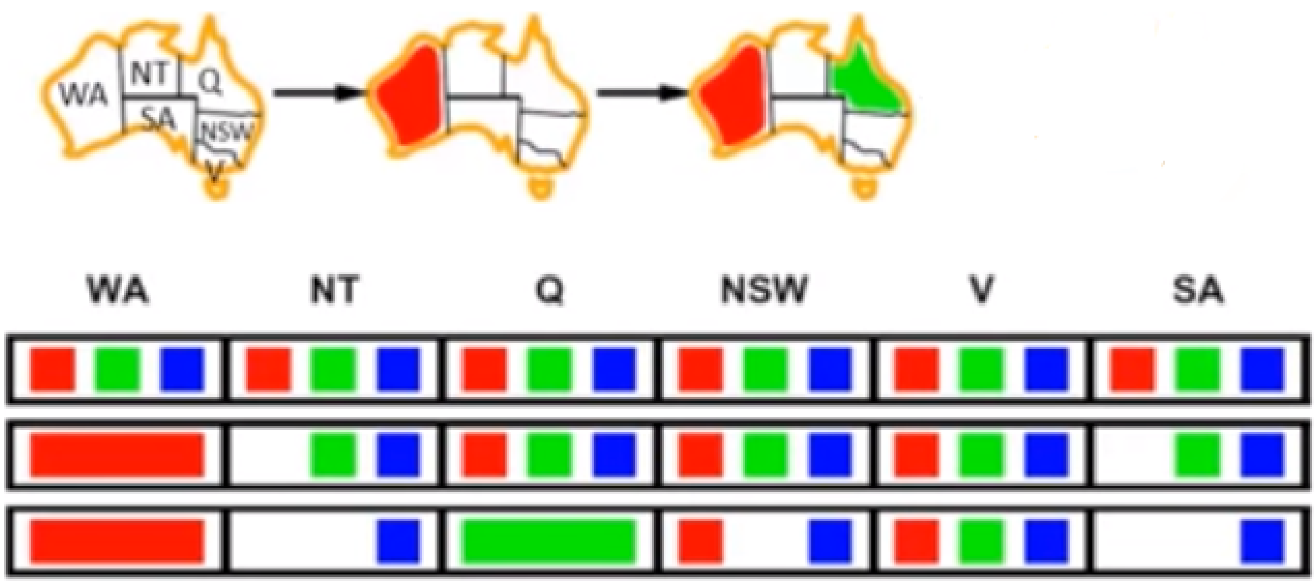
Filtering: Forward Checking

- Now, assume we color V in Blue (one of the proper values).
- We end up with that SA has no legal values.
- Sooner or later we will come to that variable, where we will backtrack. So, why not backtrack now? This saves time.



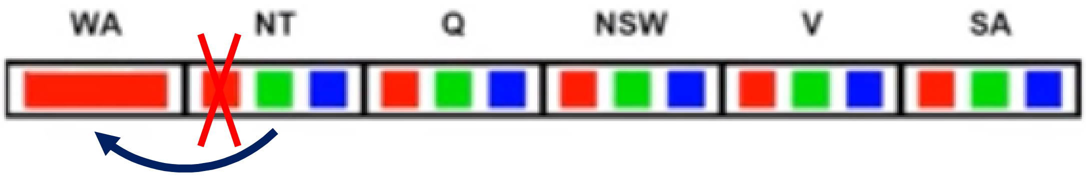
Filtering: Arc Consistency

- Forward Checking is good, but we can do better than that.
- In the following situation, we actually know NT and SA can not be Blue at the same time, which Forward Checking does not check!
- We can backtrack earlier!



Consistency of an arc

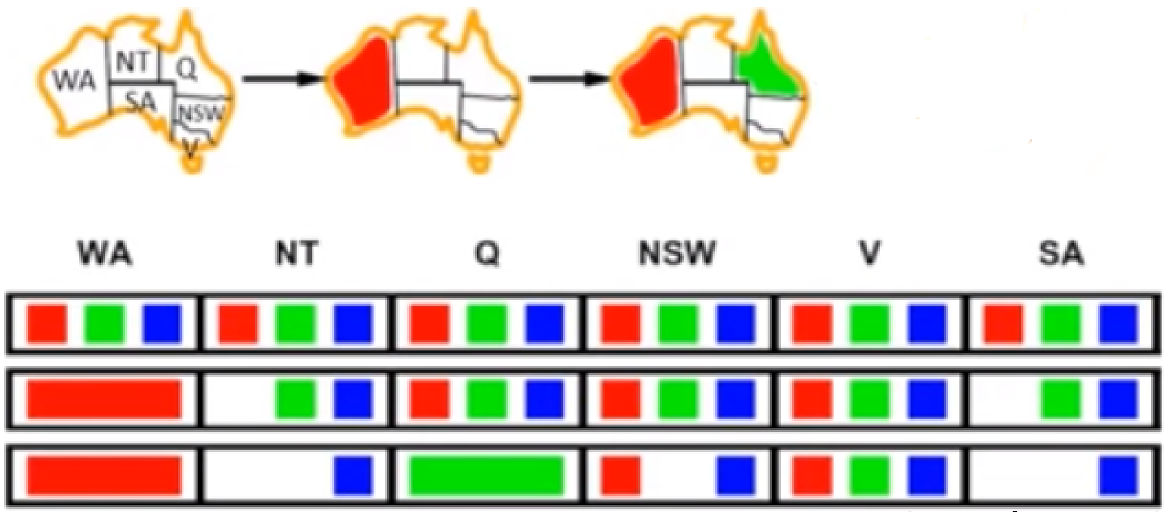
- An arc is between two variables and it has a direction (X→Y).
- An arc X→Y is ***consistent*** iff for every x in the tail (X) there is some y in the head (Y) which can be assigned without violating constraints.
- See the arc below, is it consistent?
- Red in NT domain violates! Remove it to get consistency.



- What about Q→WA? NSW→WA? ....

Arc Consistency of a CSP

- Remember, where Forward Checking left us (below figure).
- We will visit every arc, and check its consistency.

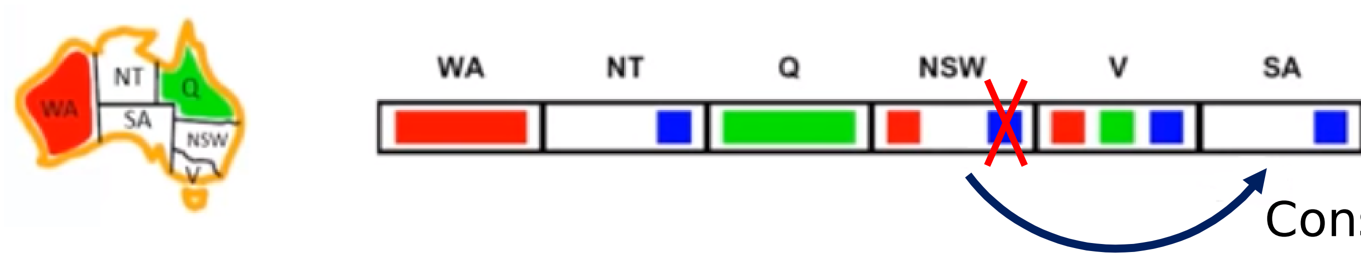


Consistent? YES

Consistent? YES

Remember: Delete from tail!

Arc Consistency of a CSP



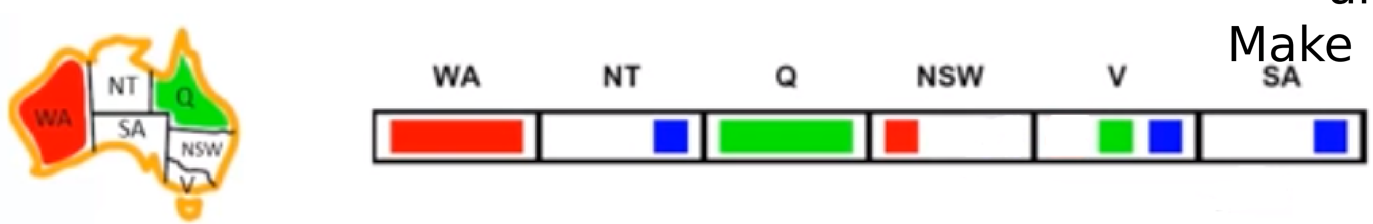
Consistent? NO

Make it consistent!

We just did this arc.
Is it still consistent? NO

When X loses a value, all incoming arcs (neighbors) are rechecked.

Make it consistent!



Remember: Delete from tail!

Filtering: Arc Consistency

- Checking over and over again makes arc consistency slower, but hopefully we will need less backtracking.
- Also, it detects failure earlier. Remember where we left:

- Arc consistency checks the arc above, detects inconsistency.
- Since no values remained in the domain, it reports failure. Algorithm backtracks!