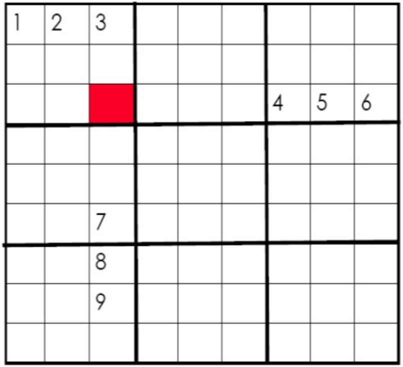
CSC384
Constraint Satisfaction Problems
Part 2

Bahar Aameri & Sonya Allin

Winter 2020



The backtracking search **won't** detect that the (3,3) cell has no possible value until all variables of the row/column/sub-square are assigned.

Constraint Propagation

- In CSPs, there might be variables that have **no possible value**, but BT **doesn't detect** this until it tries to **assign** them a value.
  This leads to the idea of Constraint Propagation (or Domain Filtering).

Constraint Propagation: "looking ahead" at the yet unassigned variables in the search, trying to detect obvious failures.
"Obvious" means things we can test/detect efficiently.

- Even if it doesn't detect an obvious failure, it might be possible to **eliminate some parts** of the future search.

Constraint Propagation

- Propagation has to be applied **during the search**; potentially at **every node** of the search tree.

- Propagation itself is an inference step that needs some resources (in particular, **time**). If propagation is slow, this can slow the search down to the point where using propagation makes finding a solution take longer!

- Two main types of propagation: **Forward Checking** and **Generalized Arc Consistency**.

Constraint Propagation: Forward Checking

Forward Checking: An extension of backtracking search.
Employs a modest amount of propagation (look ahead).

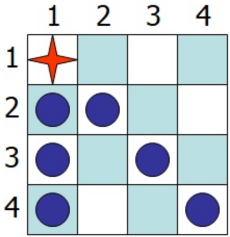
Intuition: When **instantiating** a variable V, do the following for **all constraints** C that have only **one uninstantiated** variable X remaining:

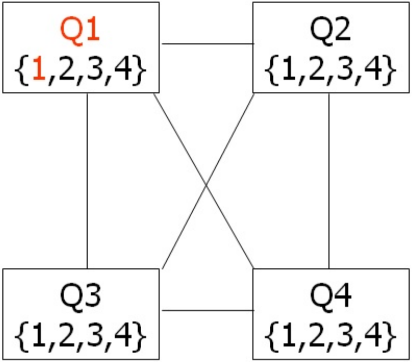
- Check all the values of X;
- Prune those values that violate C.

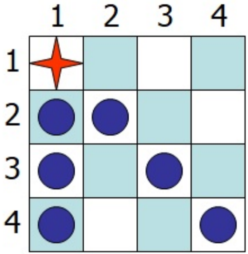
Undo the pruning when backtrack.

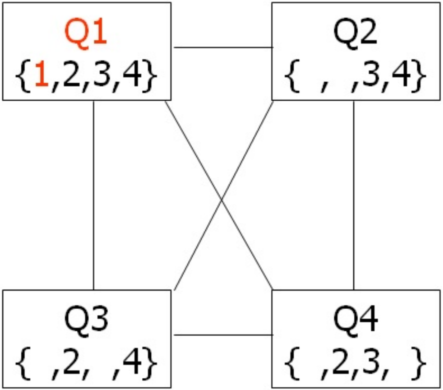
Forward Checking – Example

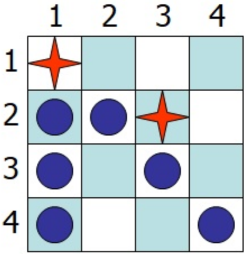
Each of $Q_1, ..., Q_4$ denotes a queen per row.
Forward checking prunes domains of $Q_1, ..., Q_4$ based on binary constraints over $Q_1, ..., Q_4$.

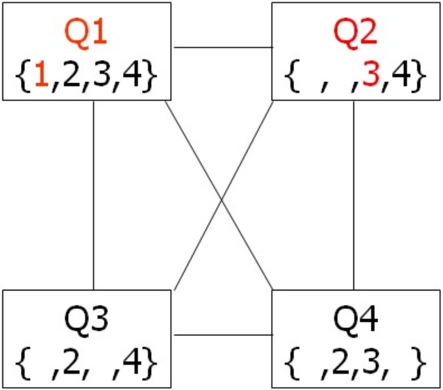


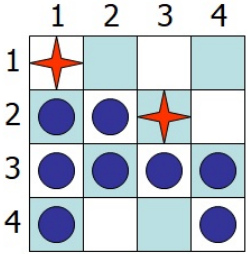


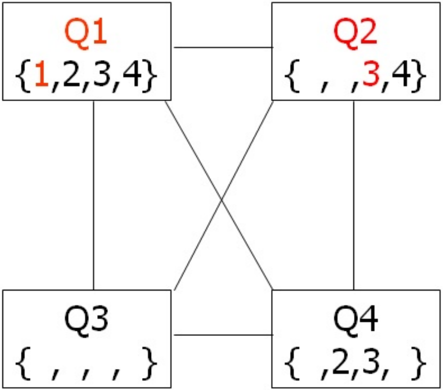

1
2
3
4
1
2
3
4

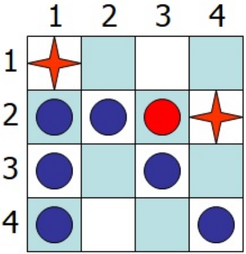

Q1
{1,2,3,4}
Q2
{ , ,3,4}
Q3
{ ,2, ,4}
Q4
{ ,2,3, }

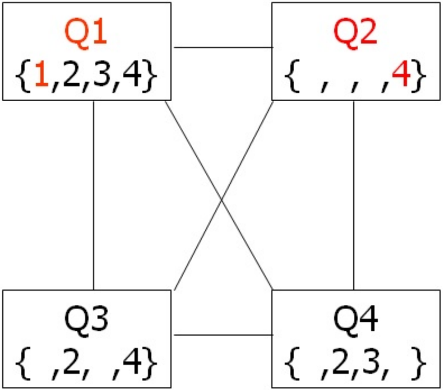

1
2
3
4
1
2
3
4

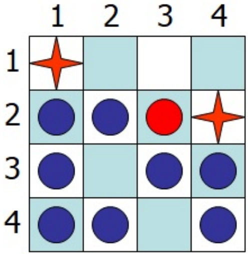

Q1
{1,2,3,4}
Q2
{ , ,3,4}
Q3
{ ,2, ,4}
Q4
{ ,2,3, }

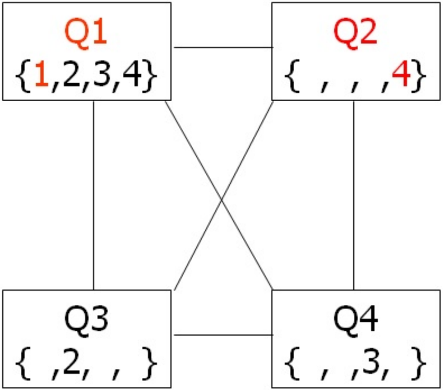

1
2
3
4
1
2
3
4

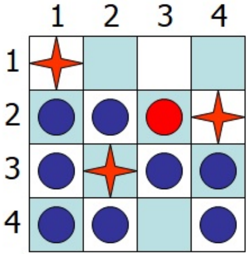

Q1
{1,2,3,4}
Q2
{ , ,3,4}
Q3
{ , , , }
Q4
{ ,2,3, }

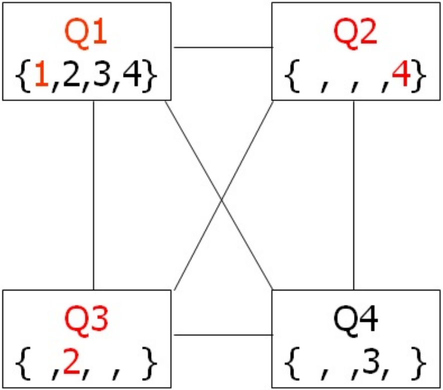

1
2
3
4
1
2
3
4

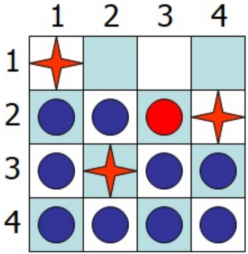

Q1
{1,2,3,4}
Q2
{ , , ,4}
Q3
{ ,2, ,4}
Q4
{ ,2,3, }

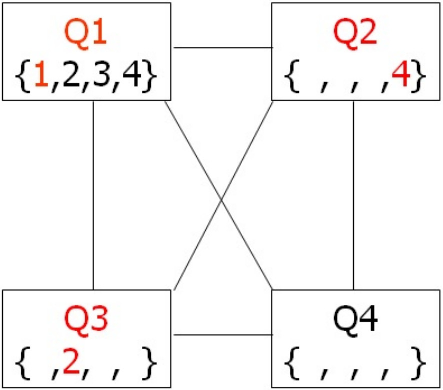

1 2 3 4
1
2
3
4

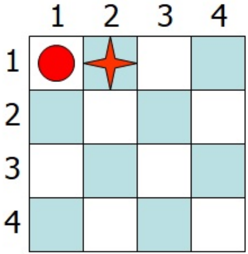

Q1
{1,2,3,4}
Q2
{ , , ,4}
Q3
{ ,2, , }
Q4
{ , ,3, }

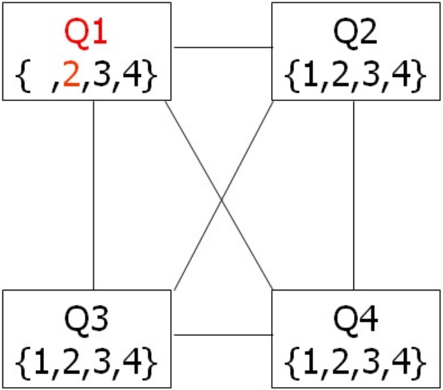

1
2
3
4
1
2
3
4

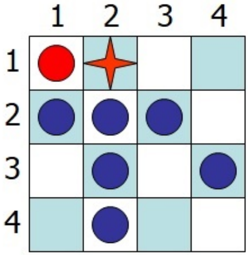

Q1
{1,2,3,4}
Q2
{ , , ,4}
Q3
{ ,2, , }
Q4
{ , ,3, }

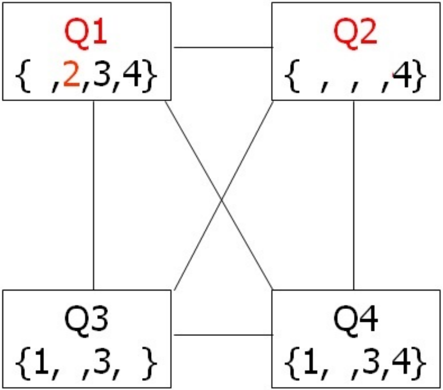

1 2 3 4
1
2
3
4

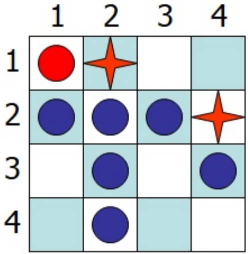

Q1
{1,2,3,4}
Q2
{ , , ,4}
Q3
{ ,2, , }
Q4
{ , , , }

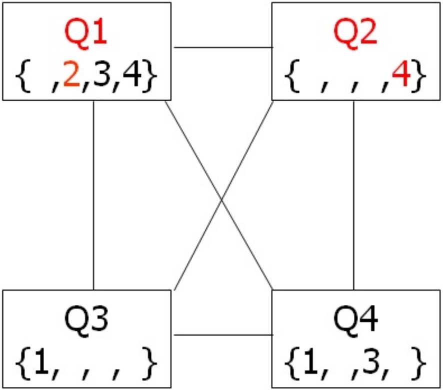

1
2
3
4
1
2
3
4

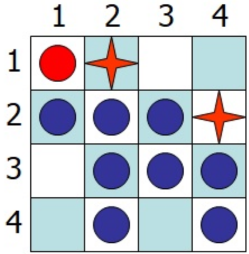

Q1
{ ,2,3,4}
Q2
{1,2,3,4}
Q3
{1,2,3,4}
Q4
{1,2,3,4}

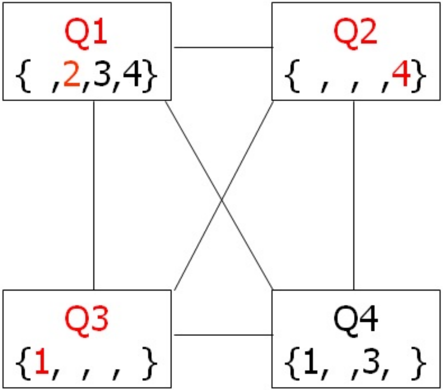

1
2
3
4
1
2
3
4

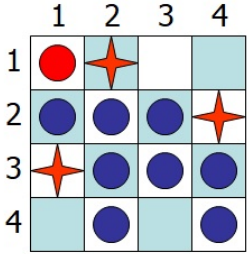

Q1
{ ,2,3,4}
Q2
{ , , ,4}
Q3
{1, ,3, }
Q4
{1, ,3,4}

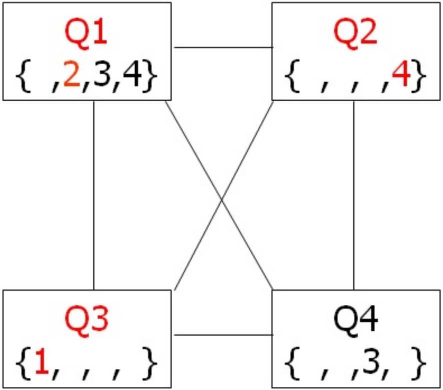

1
2
3
4
1
2
3
4

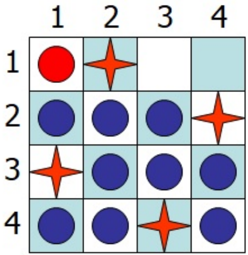

Q1
{ ,2,3,4}
Q2
{ , , ,4}
Q3
{1, , , }
Q4
{1, ,3, }

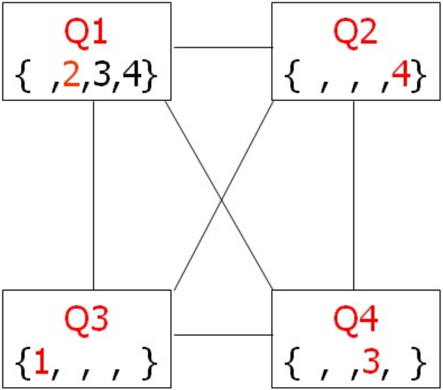

1 2 3 4
1
2
3
4


Q1
{ ,2,3,4}
Q2
{ , , ,4}
Q3
{1, , , }
Q4
{1, ,3, }


1
2
3
4
1
2
3
4


Q1
{ ,2,3,4}
Q2
{ , , ,4}
Q3
{1, , , }
Q4
{ , ,3, }


1 2 3 4
1
2
3
4


Q1
{ ,2,3,4}
Q2
{ , , ,4}
Q3
{1, , , }
Q4
{ , ,3, }

Forward Checking: The Algorithm

```
def FCCheck(C,X):
// C is a constraint with all its variables already
// assigned, except for variable X.
1.  for d := each member of CurDom(X):
2.     if making X = d together with previous assignments
                 to variables in the scope of C falsifies C:
3.        remove d from CurDom(X)
4.  if CurDom[X] == {}:
5.     RETURN DWO   # Domain Wipe Out
6.  RETURN ok
```

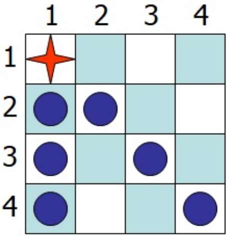
Forward Checking: The Algorithm

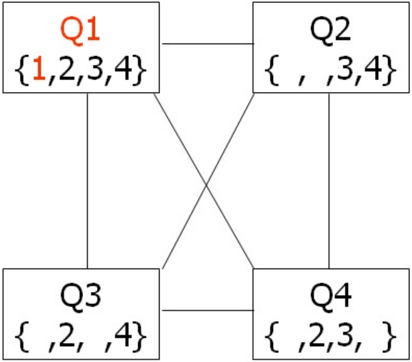
```
def FC(Level):
    if all Variables assigned
       PRINT Value of each Variable
       EXIT or RETURN                        # EXIT for only one solution
                                             # RETURN for more solutions
    V := PickUnassignedVariable()
    Assigned[V] := TRUE
    for d := each member of CurDom(V)
       Value[V] := d
       DWOoccured:= False
       for each constraint C over V such that C has only one
       unassigned variable X in its scope:
             if FCCheck(C,X) == DWO:  # X domain becomes empty
                DWOoccurred:= True
                BREAK                 # stop checking constraints
        if NOT DWOoccured:        # all constraints were ok
            FC(Level+1)
        RestoreAllValuesPrunedByFCCheck()
     Assigned[V] := FALSE     # UNDO as we have tried all of V's values
     RETURN
```

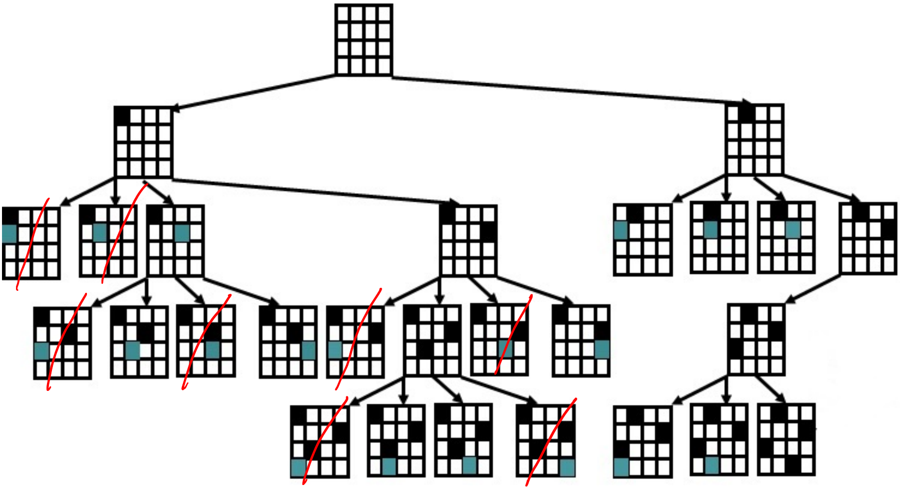
Forward Checking: Restoring Values

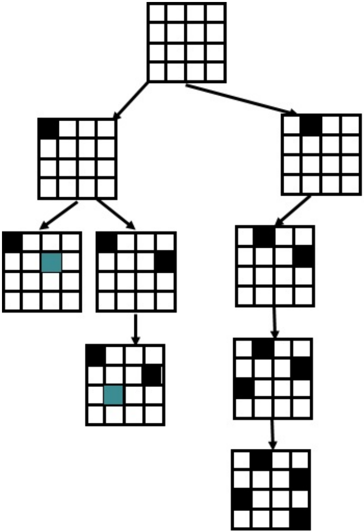
- After we **backtrack** from the current assignment the values that were pruned (as a result of that assignment) must be **restored**.

- Some **bookkeeping** needs to be done to remember which values were pruned by which assignment.


1 2 3 4
1
2
3
4


Q1
{1,2,3,4}
Q2
{ , ,3,4}
Q3
{ ,2, ,4}
Q4
{ ,2,3, }





BT vs FC: Time Efficiency

- The general class of CSPs are **NP-complete**.
  That is, their worst-case running time is **exponential**.
  BT worst-case running time: $\mathcal{O}(d^N)$, where d is the max size of a variable domain, and N is the number of variables.

- But, typically, every NP-complete family contains large sub-classes of simpler problems.

- The purpose of developing constraint propagation techniques, such as FC, is to solve those simpler sub-classes faster.

- FC **often** is about 100 times faster than BT, but it can also do worse!

More on this:

Bacchus, Fahiem, and Adam Grove. "On the forward checking algorithm." International Conference on Principles and Practice of Constraint Programming. Springer, Berlin, Heidelberg, 1995.

What variables would you try first?

$V_{1,5} = 2$

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

$\{\cancel{2}, 6\} \Rightarrow V_{9,5} = 6$

Variable and Value Ordering Heuristics

- Heuristics can be used to determine
  - the **order** in which **variables** are assigned: PickUnassignedVariable()
  - the **order** of **values** tried for each variable.
- The choice of the next variable can vary from branch to branch.
  Example: Under the assignment $V_1 = a$ we might choose to assign V_4 next, while under $V_1 = b$ we might choose to assign V_5 next.
- This dynamically chosen variable ordering has a tremendous impact on performance.

Variable and Value Ordering Heuristics

Degree Heuristic: Select the variable that is involved in the largest number of constraints on other **unassigned** variables.

Minimum Remaining Values Heuristics (MRV):

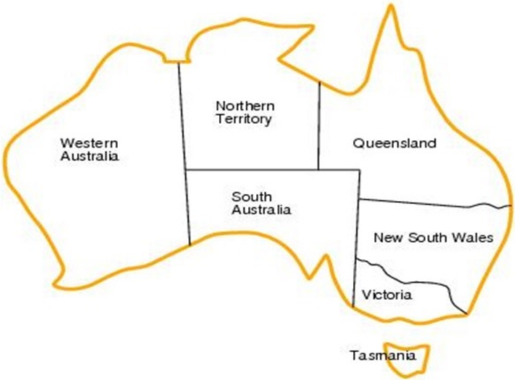
- Always branch on a variable with the smallest remaining values (smallest CurDom). **Intuition:** If a variable has only one value left, that value is forced, so we should propagate its consequences immediately.

- This heuristic tends to produce skinny trees at the top. More variables can be instantiated with fewer nodes searched.

- More constraint propagation/DWO failures occur when the tree starts to branch out. Hence, inconsistencies can be found much faster.

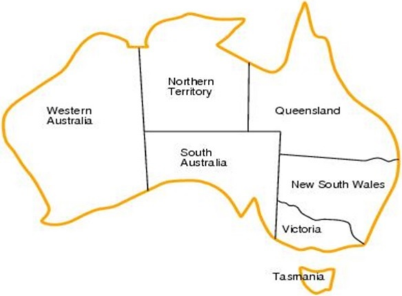
Example: Map Colouring

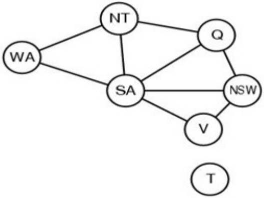
Problem Statement: Color the following map using **red**, **green**, and **blue** such that **adjacent regions have different colors**.

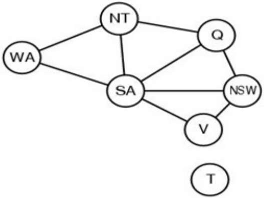


Problem formulation:

- **Variables:** WA, NT, SA, Q, NSW, V, T
- **Domains:** {R, G, B}
- **Constraints:** $WA \neq NT$, $WA \neq SA$, $SA \neq NT$, $SA \neq Q$
  $SA \neq NSW$, $SA \neq V$, $NT \neq Q$, $Q \neq NSW$
  $V \neq NSW$

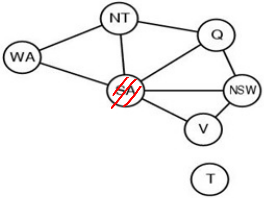


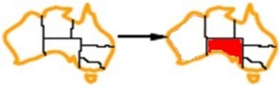



NT
Q
WA
SA
NSW
V
T

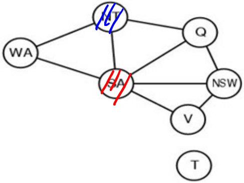

WA
NT
Q
NSW
V
SA
T

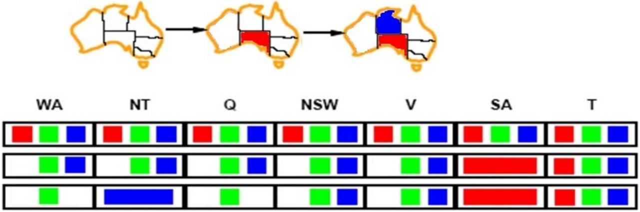
. $\{SA = red\}$ (using Degree Heuristic)





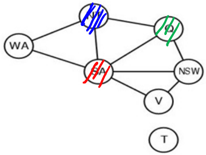
$\{SA = red, NT = blue\}$ (using MRV and Degree Heuristic results in tie between NT, Q and NSW. We choose NT).

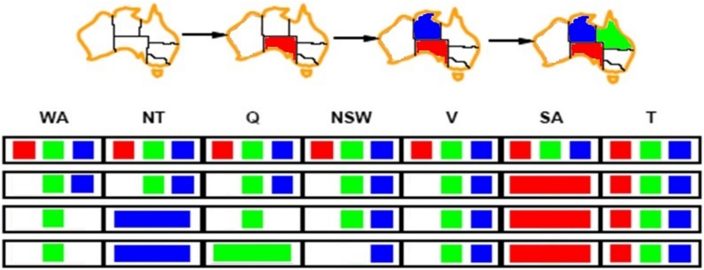




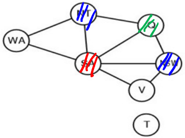
- $\{SA = red, NT = blue, Q = green\}$ (using MRV and Degree Heuristic)

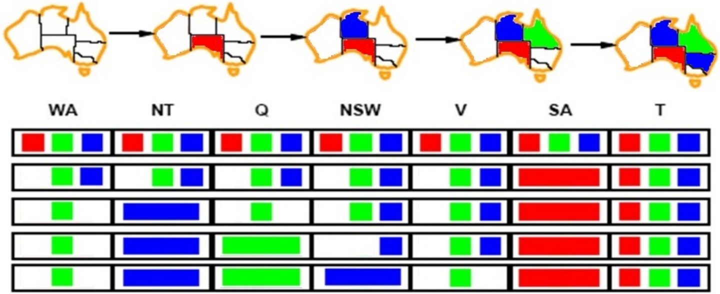
.



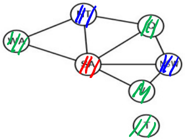


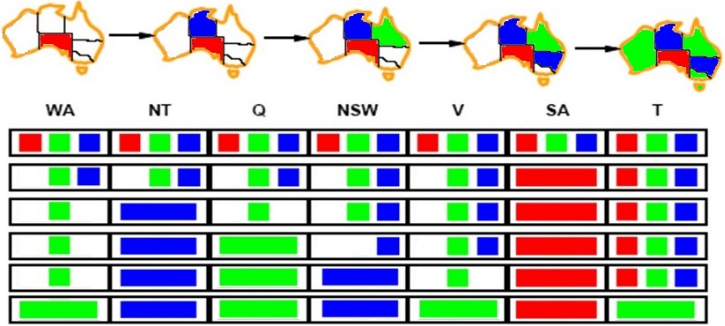
- $\{ {\color{red}SA = red}, {\color{blue}NT = blue}, {\color{green}Q = green}, {\color{blue}NWS = blue} \}$ (using MRV and Degree Heuristic)





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





Example: Map Colouring

Try the map coloring example without MRV and Degree heuristics.

FC and MRV: Empirically

- FC often is about 100 times faster than BT.

- FC with MRV (Minimal Remaining Values) often 10000 times faster.

- On some problems the speed up can be much greater.
  Converts problems that are not solvable to problems that are solvable.

- Still FC is not that powerful.
  Other more powerful forms of constraint propagation are used in practice.