Artificial Intelligence

K. N Toosi University of Technology

Course Instructor:

Dr. Omid Azarkasb

Teaching Assistants:

Atena Najafabadi Farahani

Saeed Mahmoudian

CSP

Outline

- What is a CSP
- Backtracking for CSP

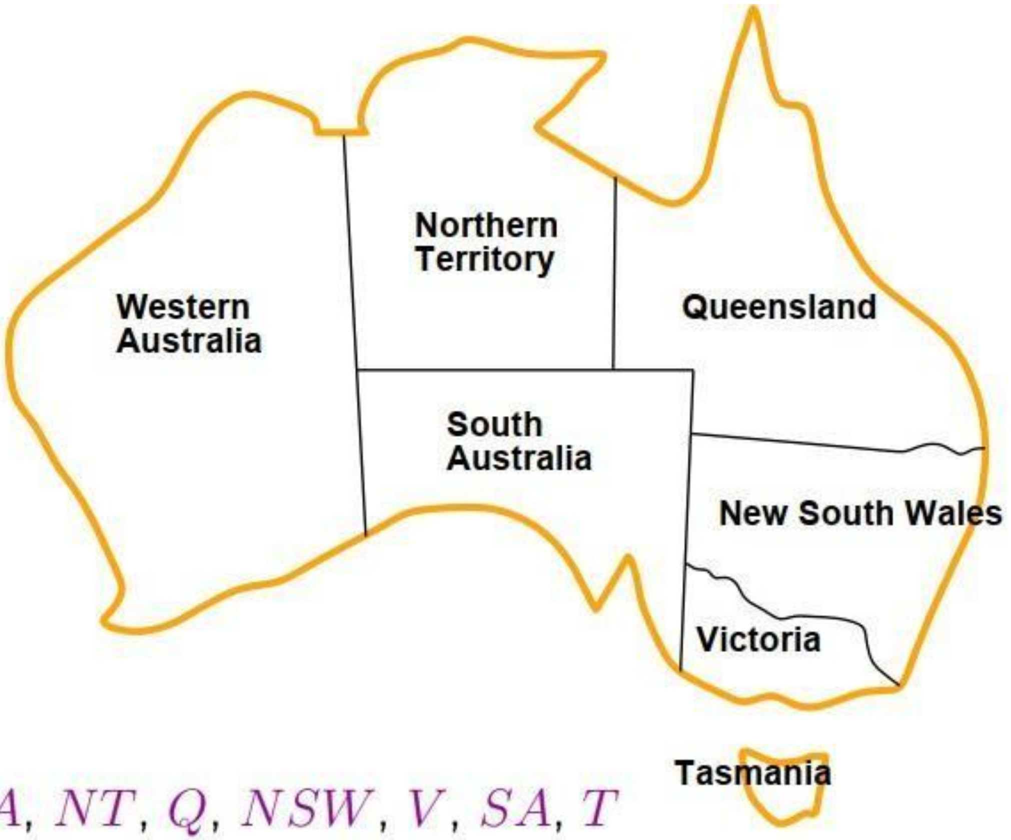
You will be Expected to Know

- Basic definitions
- Arc consistency
- Sudoku example
- Backtracking search
- Variable and value ordering: minimum-remaining values, degree heuristic, least-constraining-value
- Forward checking
- Local search for CSPs: min-conflict heuristic
- **Practical solvers**

Constraint Satisfaction Problems

- What is a CSP?
  - Finite set of variables $X_1, X_2, \ldots, X_n$
  - Nonempty domain of possible values for each variable $D_1, D_2, \ldots, D_n$
  - Finite set of constraints $C_1, C_2, \ldots, C_m$
    - Each constraint C_i limits the values that variables can take,
    - e.g., $X_1 \neq X_2$
  - Each constraint C_i is a pair **<scope, relation>**
    - Scope = Tuple of variables that participate in the constraint.
    - Relation = List of allowed combinations of variable values.
      May be an explicit list of allowed combinations.
      May be an abstract relation allowing membership testing and listing.
  - Allows useful general-purpose algorithms with more power than standard search algorithms

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., $WA \neq NT$ (if the language allows this), or

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

Example: 8-Queens

- Variables: Queens, one per column
  - Q1, Q2, …, Q8

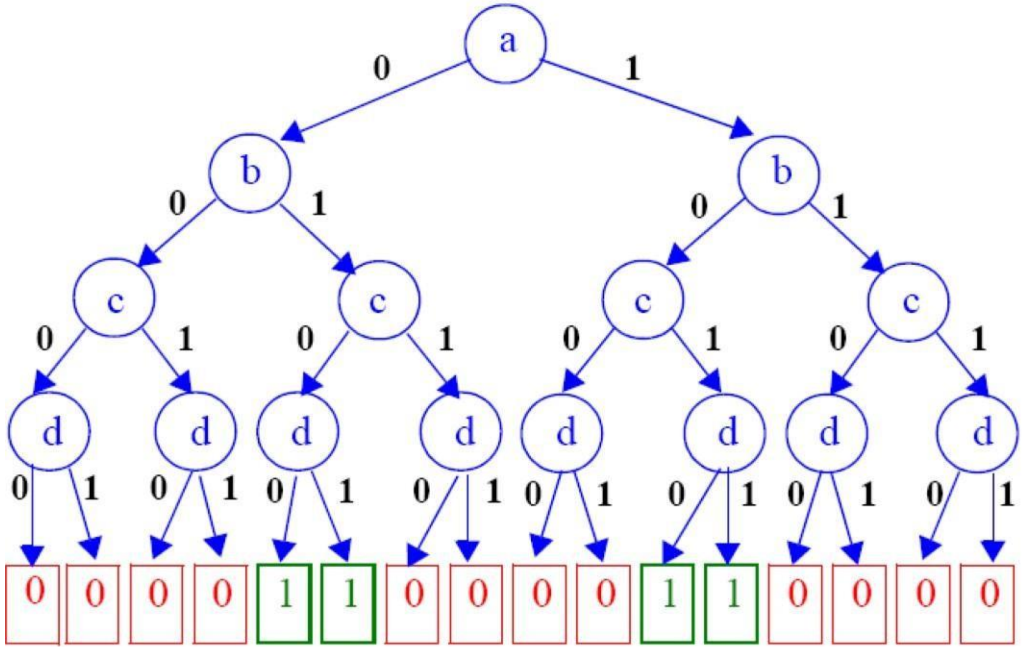
- Domains: row placement, {1, 2,…, 8}

- Constraints:
  - Qi != Qj (j != i) (cannot be in the same row)
  - |Qi – Qj| != |i – j| (cannot be in the same diagonal)

Other problems

- [Satisfiability]

$$f(a,b,c,d) = \begin{array}{l}(a \vee b \vee c)\cdot(a \vee b \vee \bar{c})\cdot(\bar{a} \vee c \vee d)\\ (\bar{a} \vee c \vee \bar{d})\cdot(\bar{b} \vee \bar{c} \vee d)\cdot(\bar{b} \vee \bar{c} \vee d)\end{array}$$



- Scheduling (Hubble telescope; class schedule; car assembly)
- Design (hardware configuration, VLSI design)

- Assignment problems
  - e.g., who teaches what class
- Timetabling problems
  - e.g., which class is offered when and where?
- Spreadsheets
- Transportation scheduling
- Factory scheduling
- Floor planning

- Notice that many real-world problems involve real-valued variables

CSPs --- what is a solution?

- A *state* is an *assignment* of values to some or all variables.
  - An assignment is ***complete*** when **every variable** has a value.
  - An assignment is ***partial*** when some variables have no values.

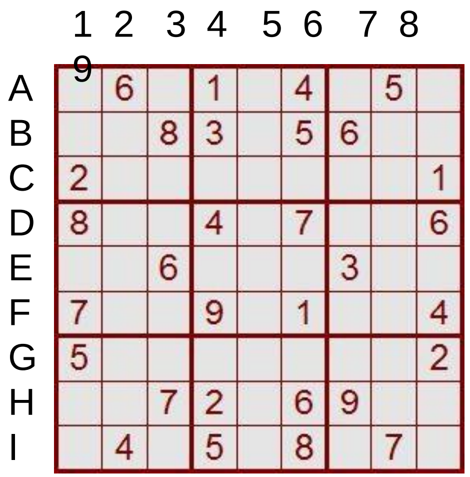
- ***Consistent assignment***
  - assignment does not violate the constraints

- A ***solution*** to a CSP is a **complete** and **consistent** assignment.

- Some CSPs require a solution that maximizes an *objective function*.

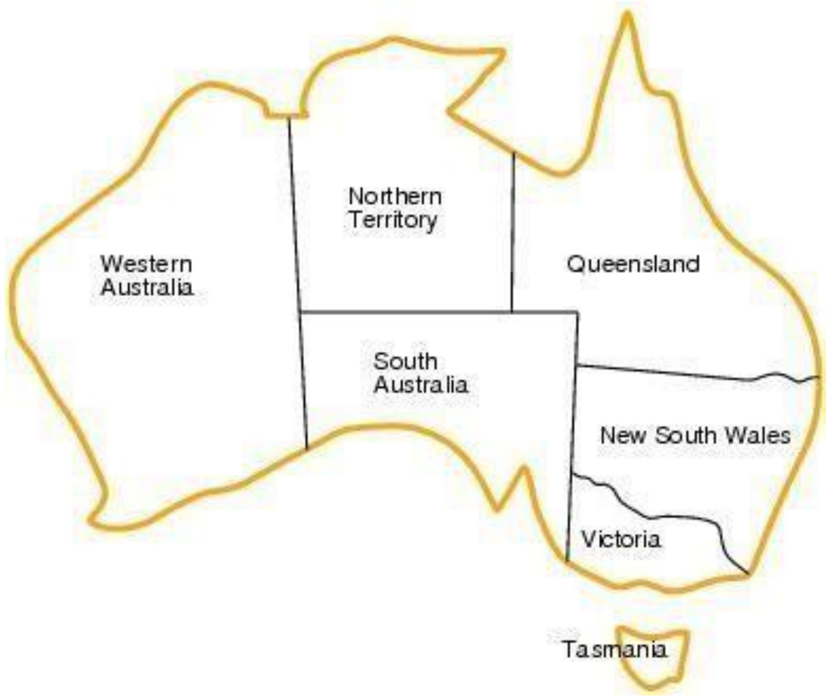
Sudoku as a Constraint Satisfaction Problem (CSP)

- Variables: 81 variables
  - A1, A2, A3, …, I7, I8, I9
  - Letters index rows, top to bottom
  - Digits index columns, left to right
- Domains: The nine positive digits
  - A1 ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9}
  - Etc.
- Constraints: 27 *Alldiff* constraints
  - *Alldiff*(A1, A2, A3, A4, A5, A6, A7, A8, A9)
  - Etc.
- [Why constraint satisfaction?]



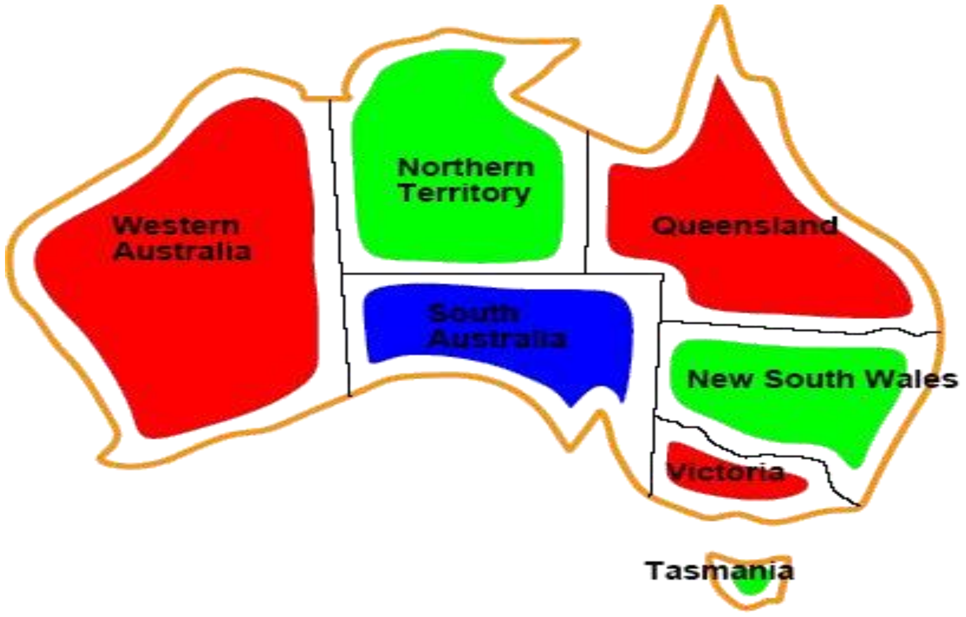
	1	2	3	4	5	6	7	8	9
A		6		1		4		5	
B			8	3		5	6		
C	2								1
D	8			4		7			6
E			6				3		
F	7			9		1			4
G	5								2
H			7	2		6	9		
I		4		5		8		7	

CSP 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. $WA \neq NT$

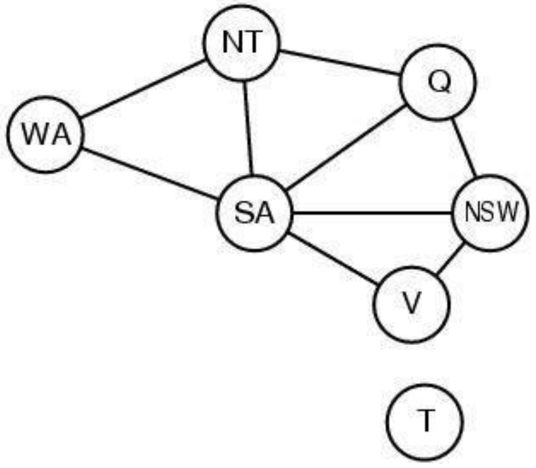
CSP example: map coloring



- Solutions are assignments satisfying all constraints, e.g. *{WA=red,NT=green,Q=red,NSW=green,V=red,SA=blue,T=green}*

Constraint graphs

- Constraint graph:
  - nodes are variables
  - arcs are binary constraints



- Binary CSP: each constraint relates at most two variables

- Graph can be used to simplify search

  e.g. Tasmania is an independent subproblem

Varieties of constraints

- Unary constraints involve a single variable.
  - e.g. $SA \neq green$
- Binary constraints involve pairs of variables.
  - e.g. $SA \neq WA$
- Higher-order constraints involve 3 or more variables.
  - Professors A, B, and C cannot be on a committee together
  - Can always be represented by multiple binary constraints
- Preference (soft constraints)
  - e.g. *red* is better than *green* often can be represented by a cost for each variable assignment
  - Combination of **optimization** with **CSPs**

Varieties of CSPs

Discrete variables

- finite domains; size d $\Rightarrow$ $O(d^n)$ complete assignments
  - ◇ e.g., Boolean CSPs, incl. Boolean satisfiability (NP-complete)
- infinite domains (integers, strings, etc.)
  - ◇ e.g., job scheduling, variables are start/end days for each job
  - ◇ need a constraint language, e.g., $StartJob_1 + 5 \leq StartJob_3$
  - ◇ linear constraints solvable, nonlinear undecidable

Continuous variables

- ◇ e.g., start/end times for Hubble Telescope observations
- ◇ linear constraints solvable in poly time by LP methods

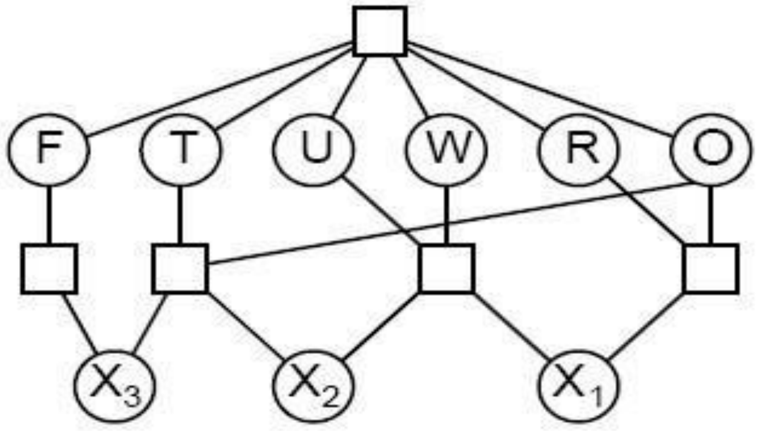
CSP Example: Cryptharithmetic puzzle

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

Variables: $F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$
Domains: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
Constraints
    alldiff(F, T, U, W, R, O)
    $O + O = R + 10 \cdot X_1$, etc.

CSP Example: Cryptharithmetic puzzle

```
    T W O
+   T W O
---------
  F O U R
```



Variables: $F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$

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

Constraints

alldiff(F,T,U,W,R,O)

$O + O = R + 10 \cdot X_1$, etc.

CSP Example: Cryptharithmetic puzzle

- Find numeric substitutions that make an equation hold:

```
      T  W  O
 +    T  W  O
 = F  O  U  R
```

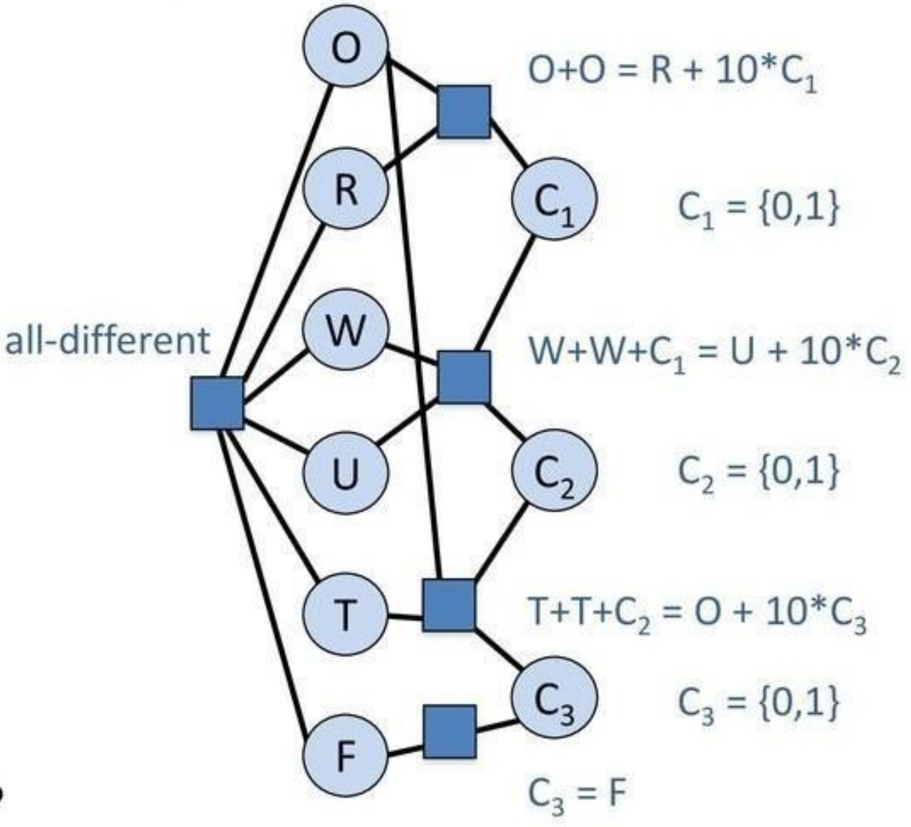
For example:

O	=	4
R	=	8
W	=	3
U	=	6
T	=	7
F	=	1

```
      7  3  4
 +    7  3  4
 = 1  4  6  8
```

Note: not unique – how many solutions?

Non-pairwise CSP:



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 an unassigned variable provided that it does not violate a constraint
  - *Goal test*: the current assignment is complete (by construction it is consistent)
  - *Path cost*: constant cost for every step (not really relevant)
- Can also use complete-state formulation
  - Local search techniques (Chapter 4)

CSP as a standard search problem

- Solution is found at depth *n* (if there are *n* variables).

- Consider using BrFS
  - Number of children of the start node is *nd*
  - Each of those has (n-1)d
  - ….

- End up with $n!d^n$ leaves even though there are only d^n complete assignments!

Commutativity

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

  → there are d^n leaves

  (will need to figure out later which variable to assign a value to at each node)

Backtracking search

- **Depth-first search** in the context of CSP is also called “**backtracking**”
- Chooses values for one variable at a time and backtracks when a variable has no legal values left to assign.
- Nice: we have a standard representation.
- No need for a domain-specific initial state, successor function, or goal test.

Backtracking search

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

function BACKTRACK(assignment, csp) return a solution or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment then
            add {var=value} to assignment
            inferences ← INFERENCE(csp, assignment)
            if inferences != failure
            add inferences to assignment
            result ← BACTRACK(assignment, csp)
                if result ≠ failure  then return result
        remove {var=value} and inferences from assignment (if you added it)
    return failure
```

Improving CSP efficiency

- Previous improvements on uninformed search
  → introduce heuristics

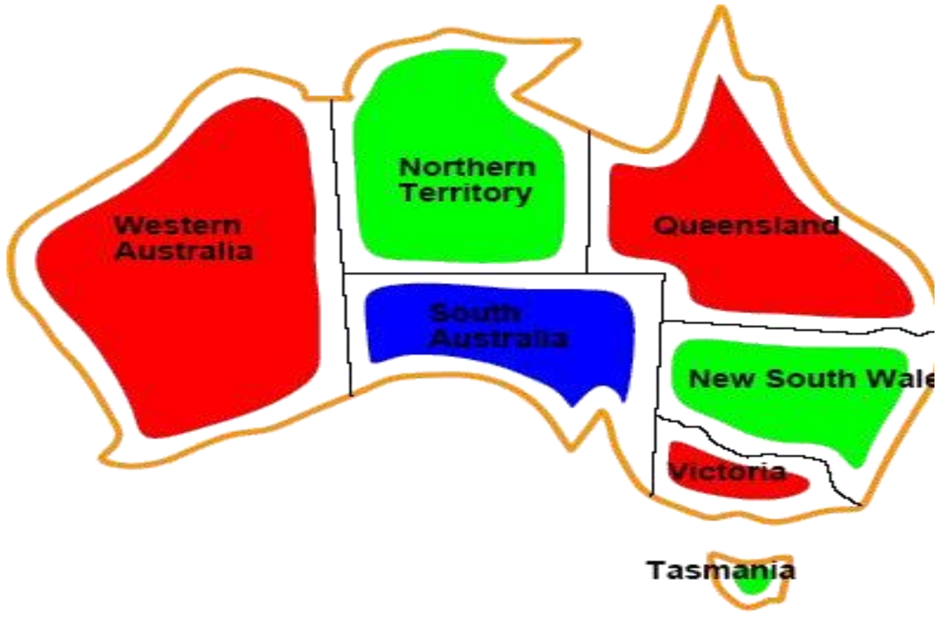
- For CSPs, general-purpose methods can give large gains in speed, e.g.,
  - Which variable should be assigned next?
  - In what order should its values be tried?
  - Can we detect inevitable failure early?

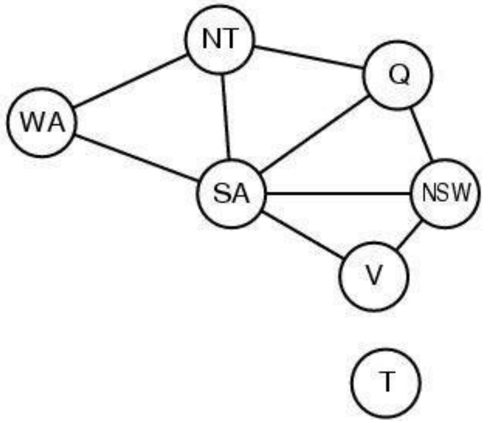
Backtracking search

SELECT-UNASSIGNED-VARIABLE

- Minimum Remaining Values (**MRV**)
  - Most constrained variable
  - Most likely to fail soon (so prunes pointless searches)
- If a tie (such as choosing the start state), choose the variable involved in the most constraints (**degree heuristic**)
- E.g., in the map example, SA adjacent to the most states.
  - Reduces branching factor, since fewer legal successors of that node

CSP example: map coloring





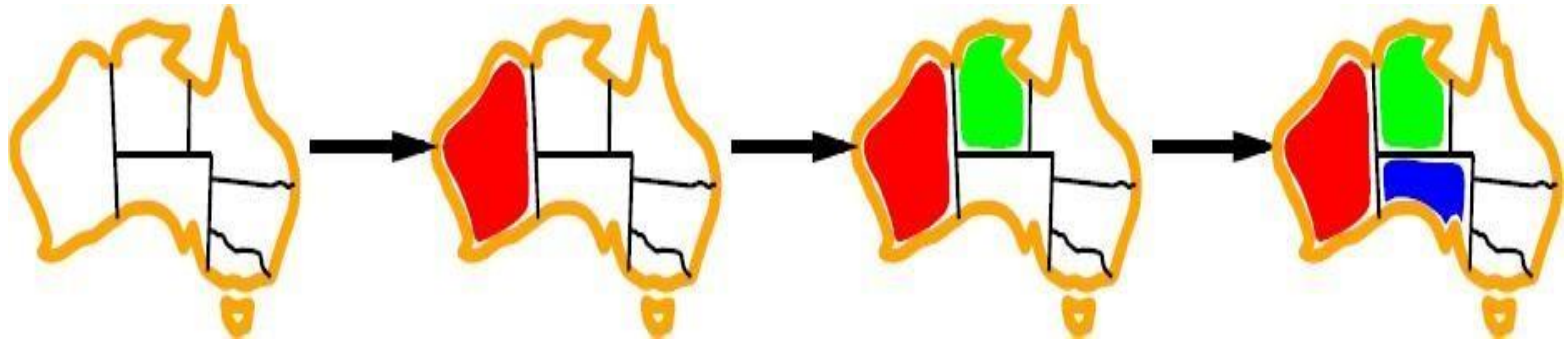
- Solutions are assignments satisfying all constraints, e.g. *{WA=red,NT=green,Q=red,NSW=green,V=red,SA=blue,T=green}*

Backtracking search

ORDER-DOMAIN-VALUES

- **Least Constraining Value**
- Rules out the fewest choices for the variables it is in constraints with
- Leave the maximum flexibility
- You have chosen the variable, now let’s make the most of it

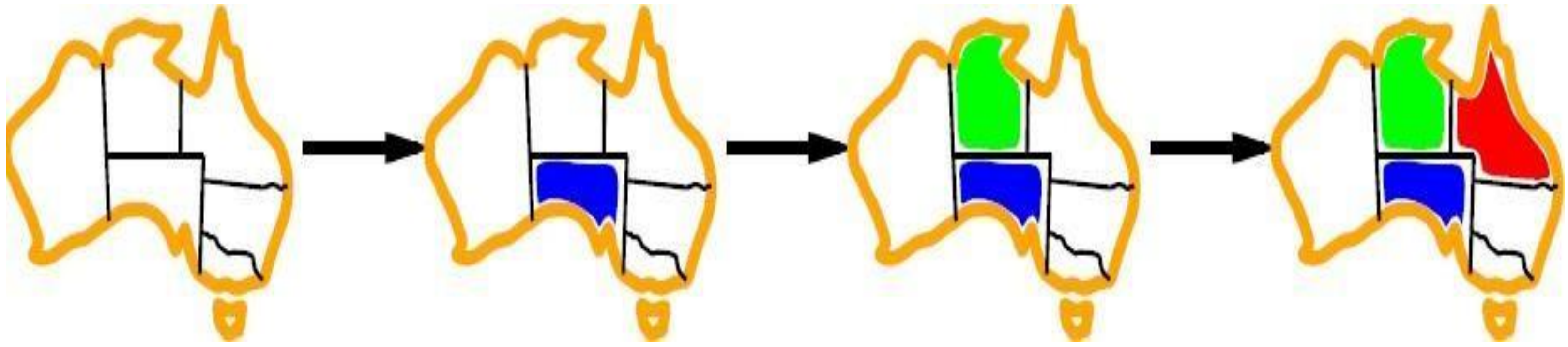
Minimum remaining values (MRV)



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

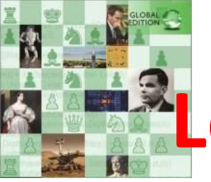
- Before the assignment to the rightmost state: one region has one remaining; one region has two; three regions have three.
- Choose the region with only one remaining

Degree heuristic for resolving ties among variables

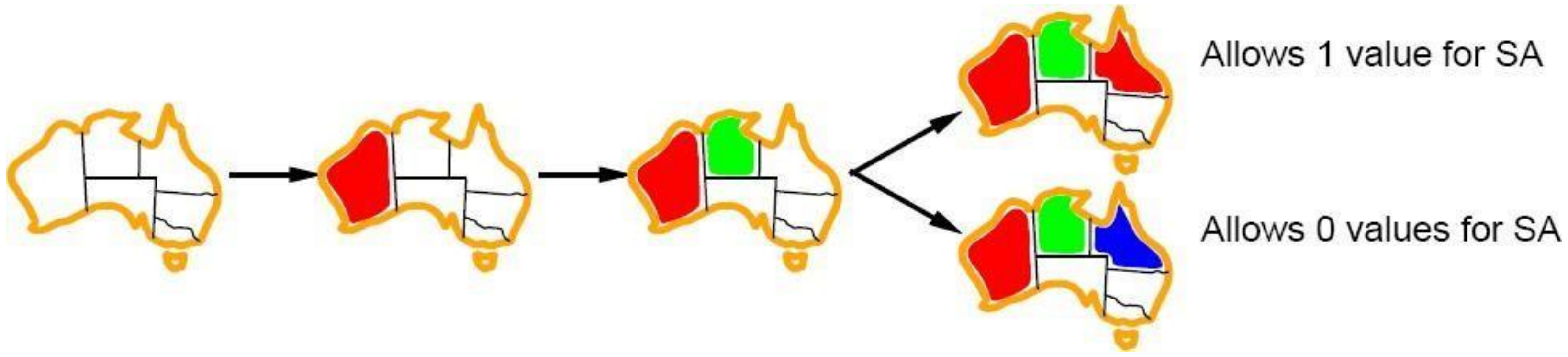


- Degree heuristic can be useful as a tie breaker.

- Before the assignment to the rightmost state, WA and Q have the same number of remaining values ({R}).
- So, choose the one adjacent to the most states. This will cut down on the number of legal successor states to it.



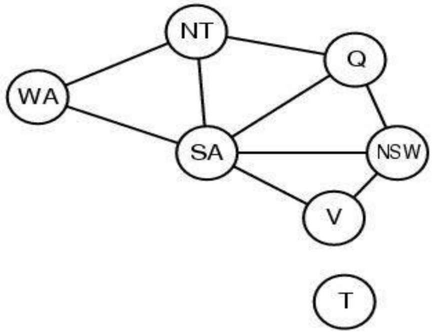
Least constraining value for value-ordering

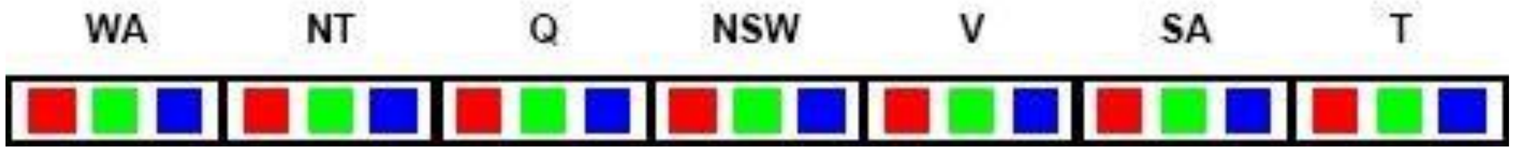


- Least constraining value heuristic

- Heuristic Rule: given a variable choose the least constraining value
  - leaves the maximum flexibility for subsequent variable assignments

Forward checking (INFERENCE)



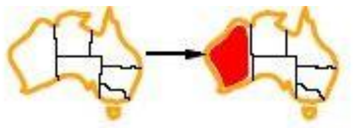


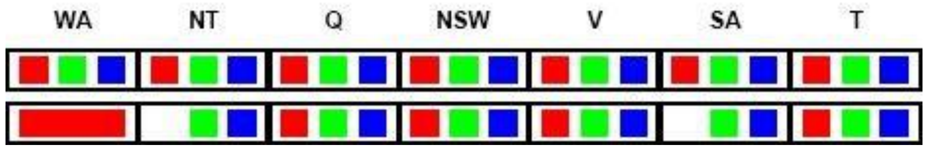
- Can we detect inevitable failure early?
  - *And avoid it later?*

- *Forward checking idea:* **keep track of remaining legal values for unassigned variables.**

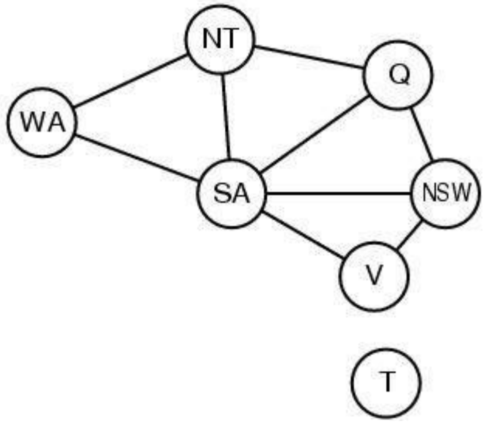
- Terminate search when any variable has no legal values.

Forward checking





WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦

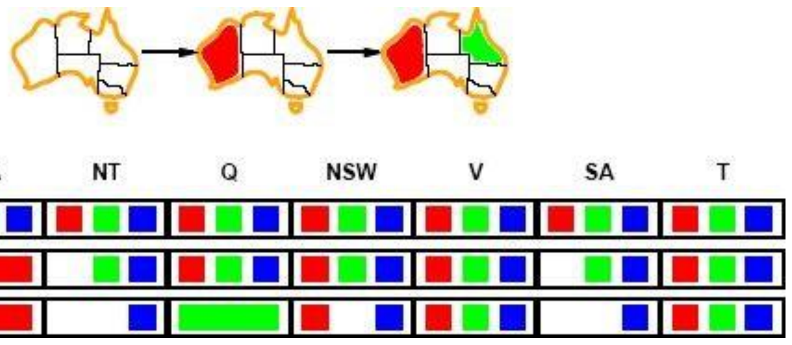


- Assign *{WA=red}*

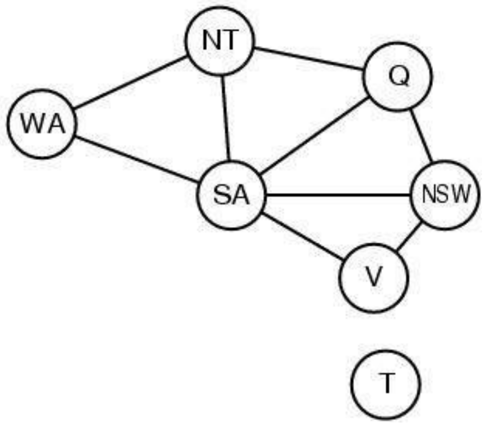
- Effects on other variables connected by constraints to WA
  - *NT can no longer be red*
  - *SA can no longer be red*

- *Note: this example is not using MRV; if it were, we would choose NT or SA next. But, we will choose Q next. This example is from the text. It shows the example here, then talks through what would happen if we had used MRV.*

Forward checking



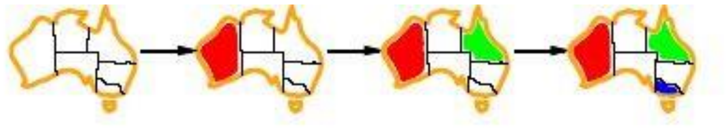
WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦
🟥	🟦	🟩	🟥🟦	🟥🟩🟦	🟦	🟥🟩🟦

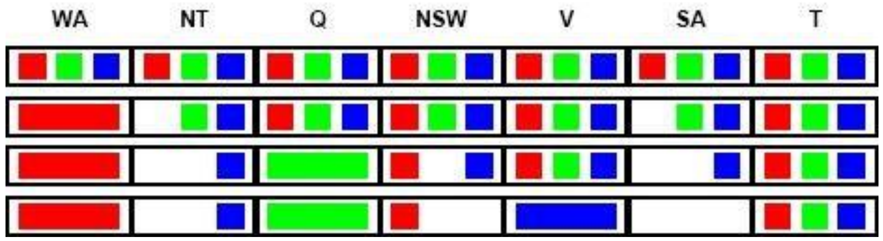


- Assign *{Q=green}*

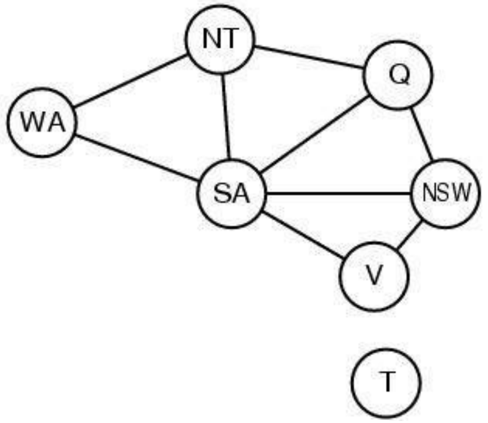
- Effects on other variables connected by constraints with WA
  - *NT can no longer be green*
  - *NSW can no longer be green*
  - *SA can no longer be green*

Forward checking





WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦
🟥	🟦	🟩	🟥🟦	🟥🟩🟦	🟦	🟥🟩🟦
🟥	🟦	🟩	🟥	🟦		🟥🟩🟦

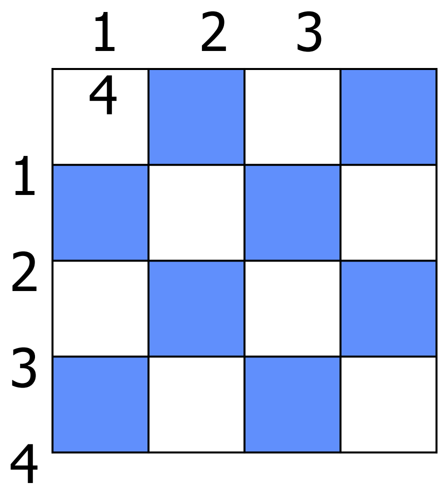


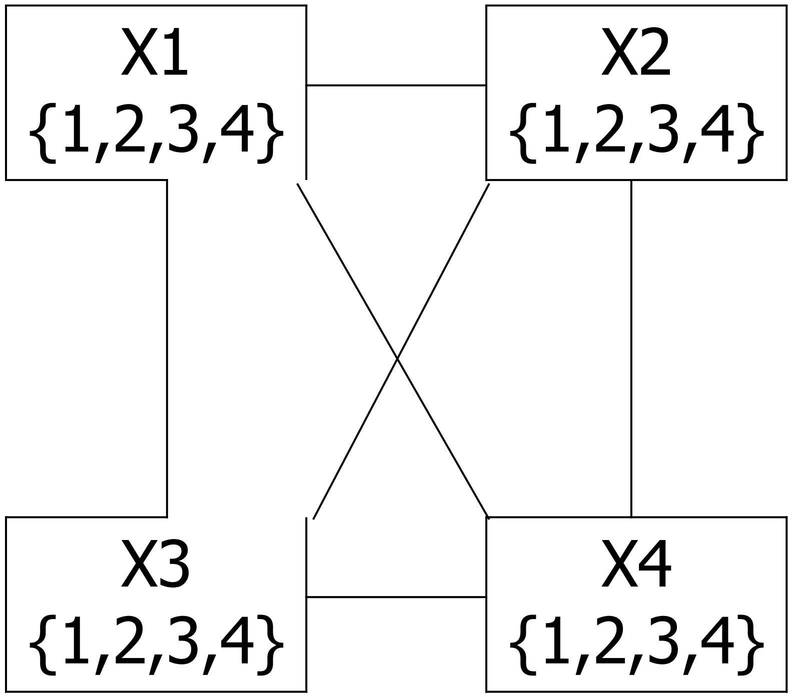
- Assign *{V=blue}*
- Effects on other variables connected by constraints with WA
  - *NSW can no longer be blue*
  - *SA is empty*
- Forward Checking has detected that partial assignment is *inconsistent* with the constraints and backtracking can occur.

Backtracking search

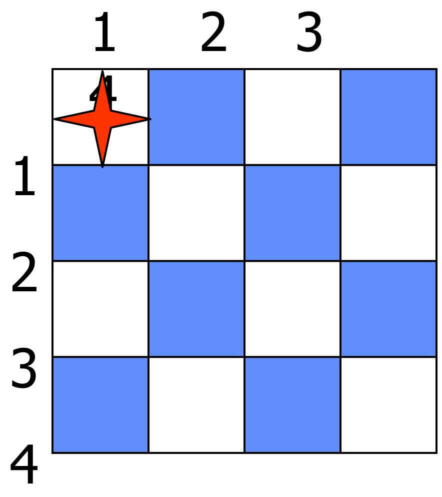
```
function BACKTRACKING-SEARCH(csp) return a solution or failure
    return BACKTRACK({} , csp)

function BACKTRACK(assignment, csp) return a solution or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(csp,assignment)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
          if value is consistent with assignment then
                     add {var=value} to assignment
             inferences ← INFERENCE(csp,assignment)
                     if inferences != failure
               add inferences to assignment
               result ← BACTRACK(assignment, csp)
                       if result ≠ failure then return result
          remove {var=value} and inferences from assignment (if you added it)
    return failure
```

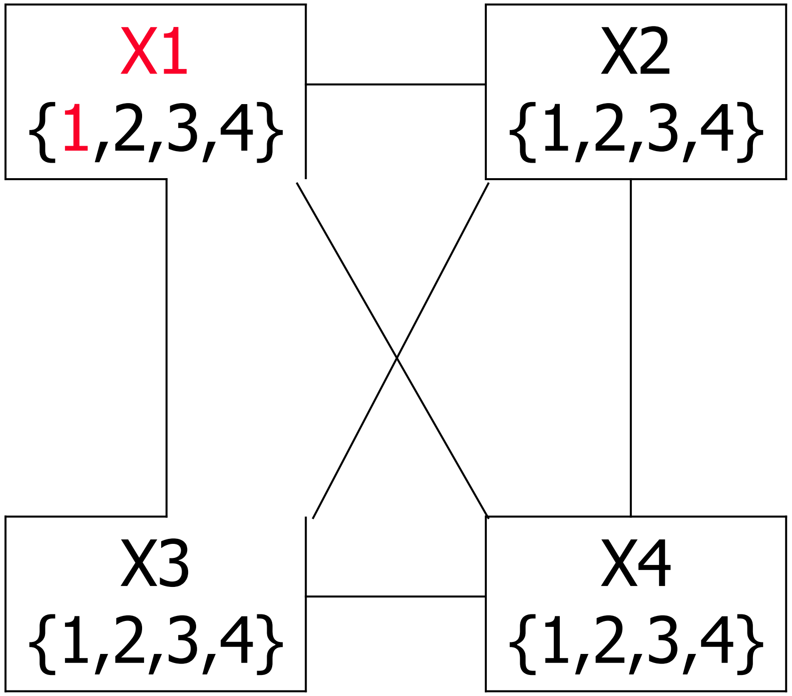
Example: 4-Queens Problem



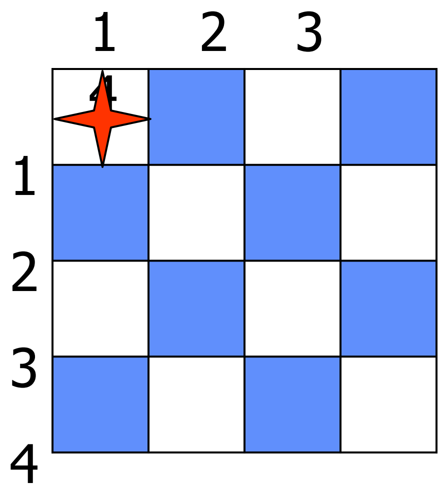


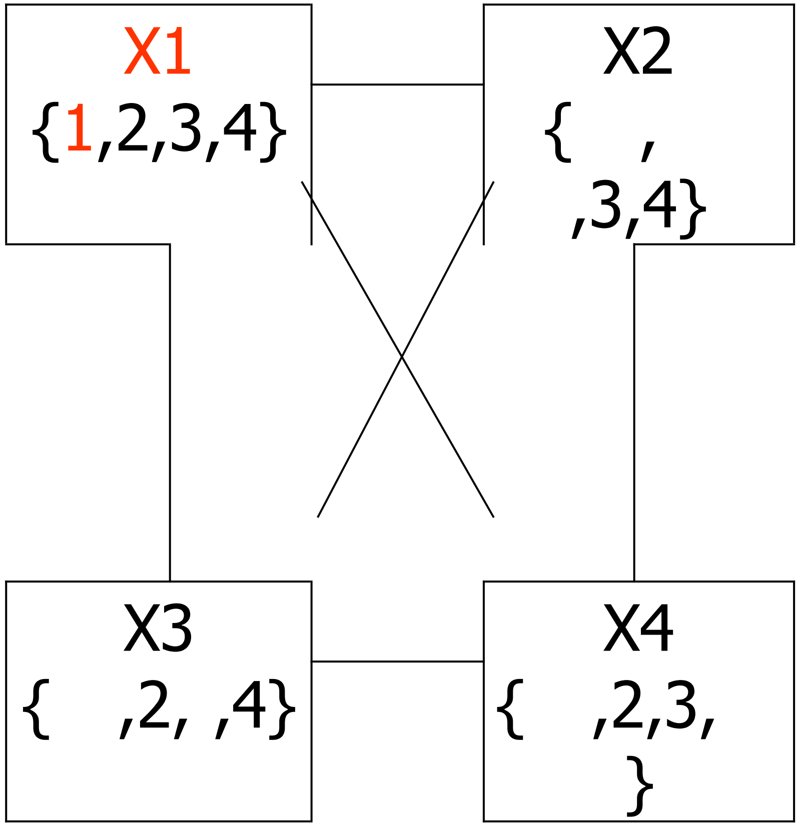
Example: 4-Queens Problem



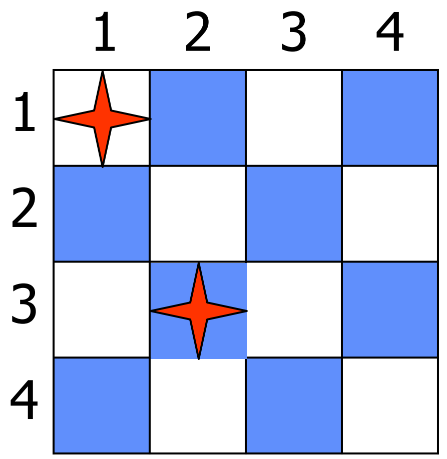


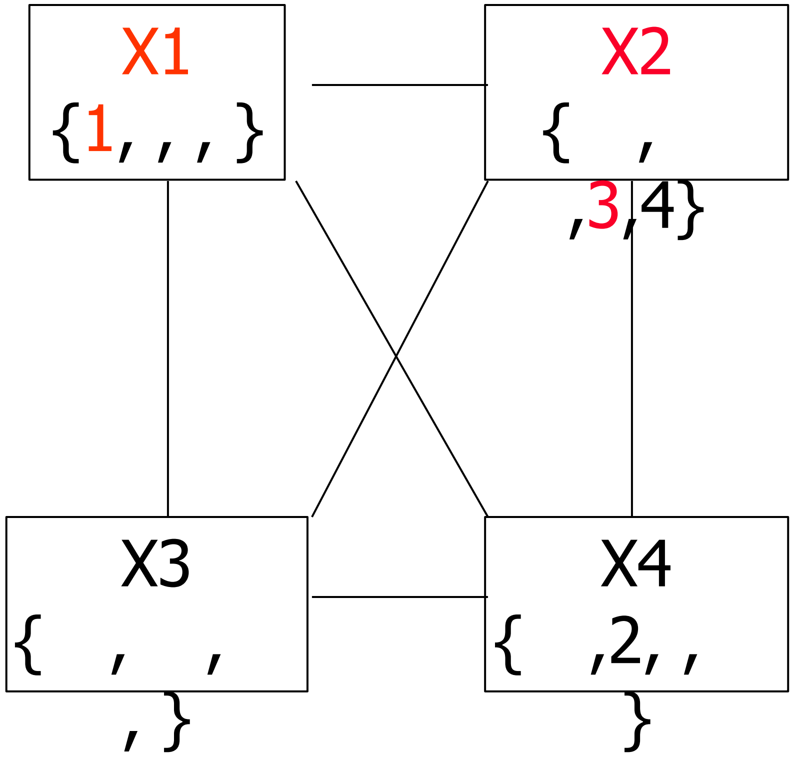
Example: 4-Queens Problem



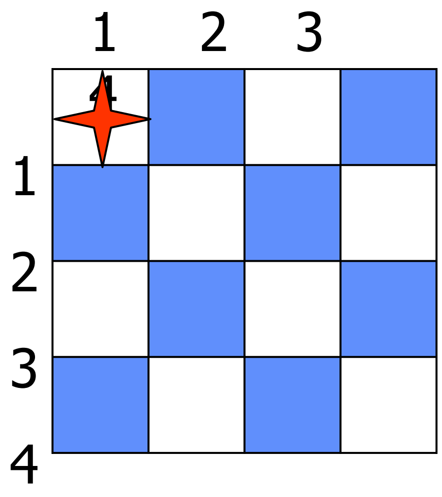


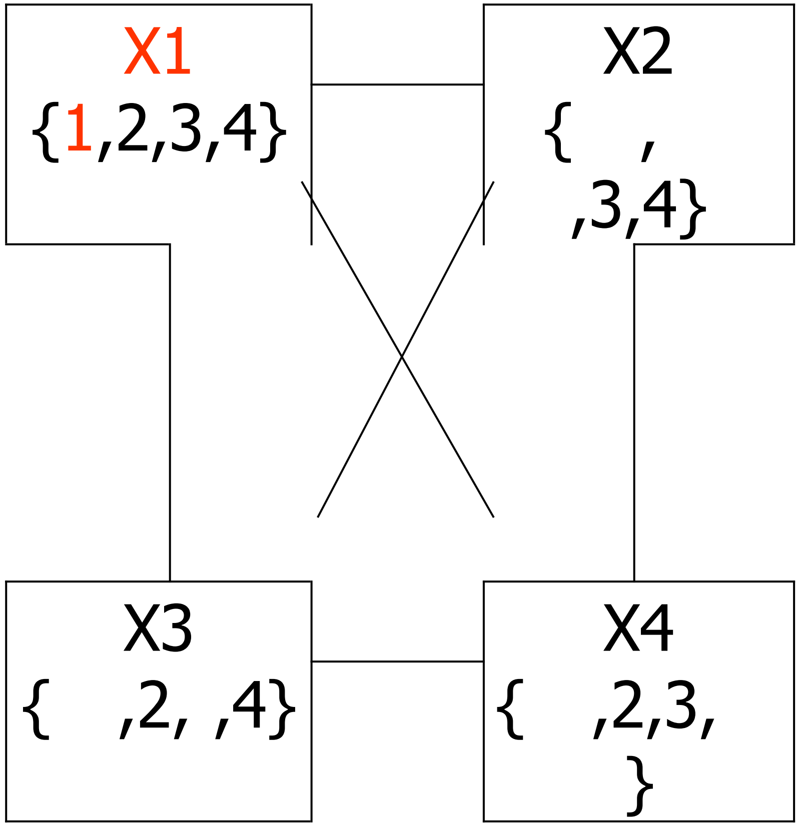
Example: 4-Queens Problem



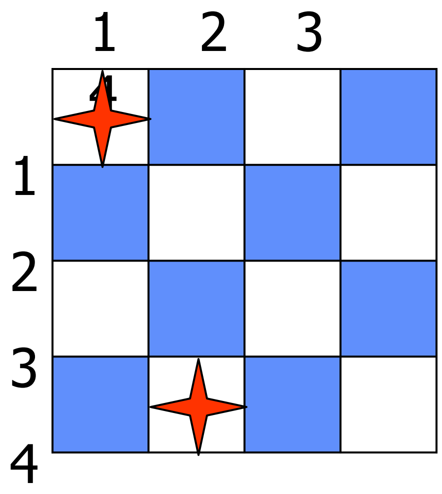


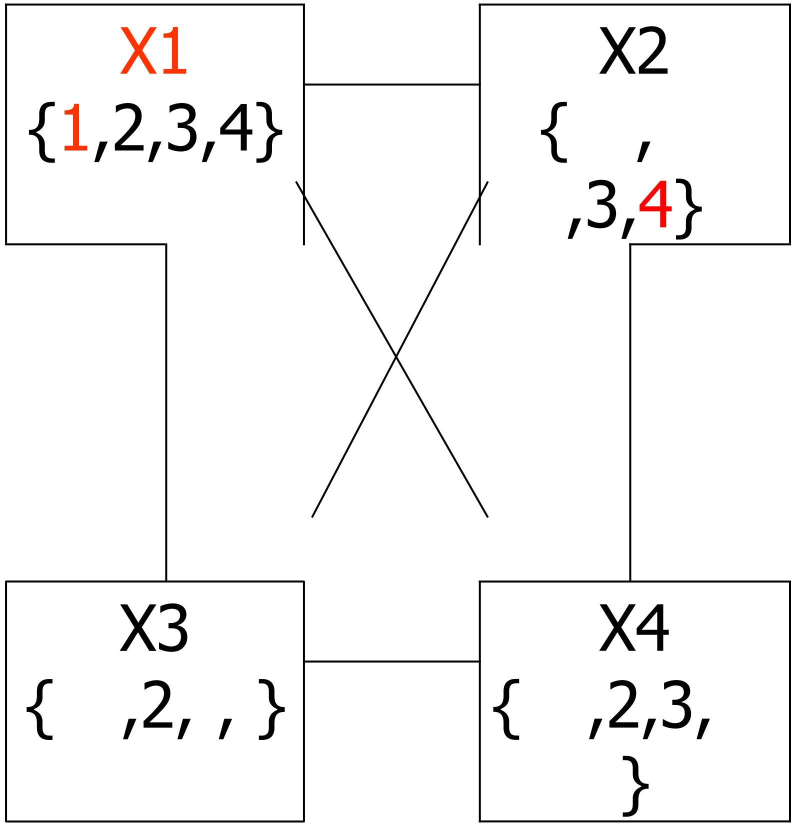
Example: 4-Queens Problem



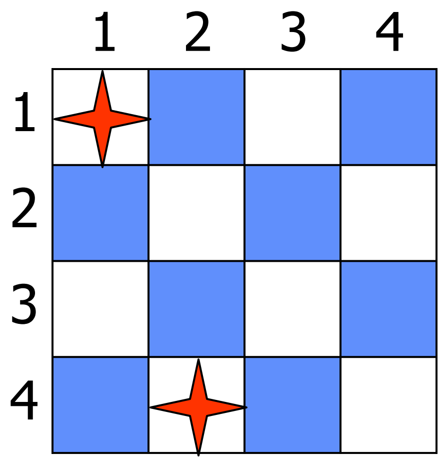


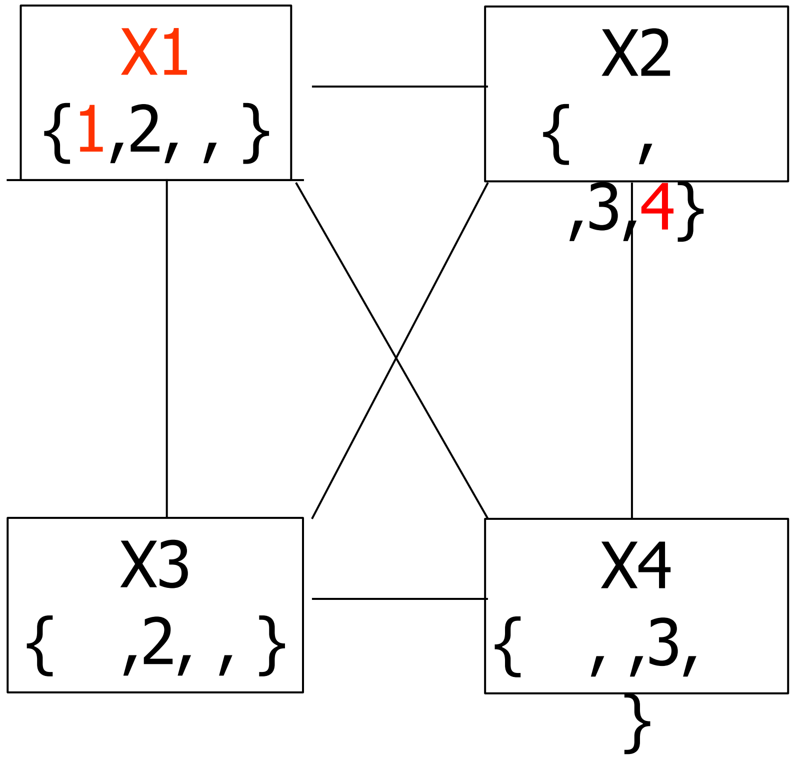
Example: 4-Queens Problem



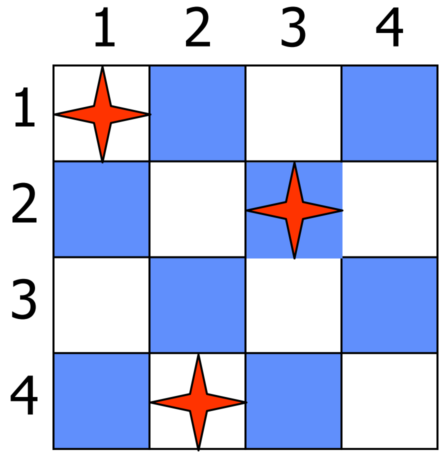


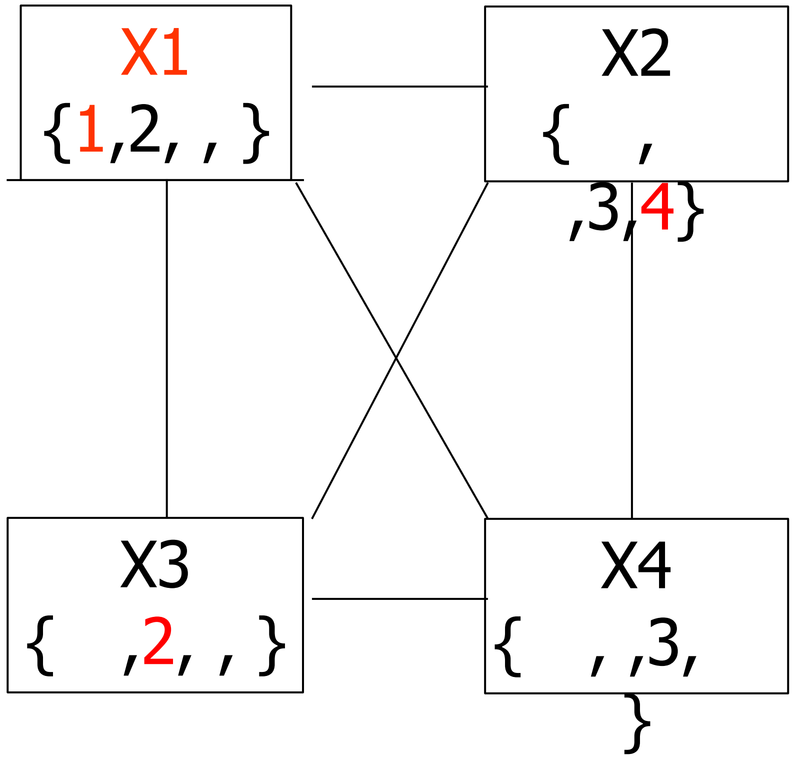
Example: 4-Queens Problem



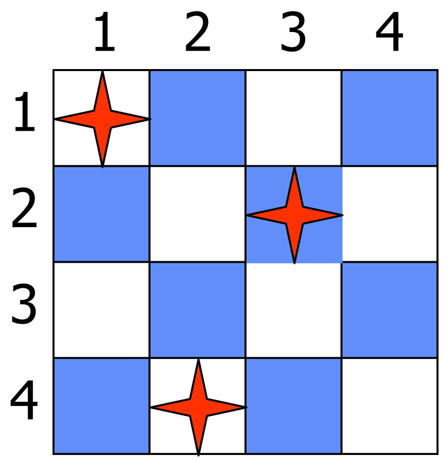


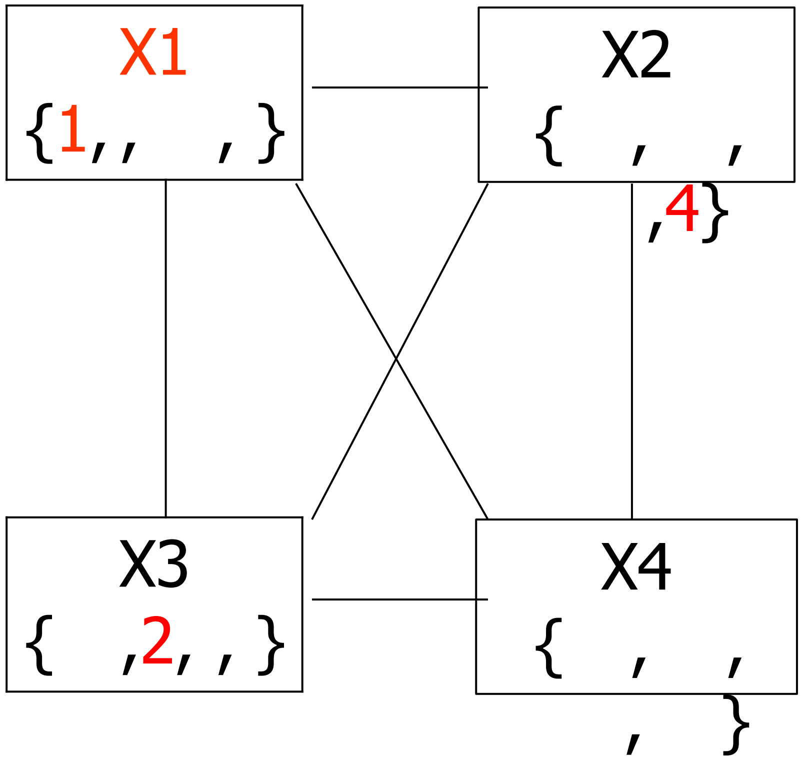
Example: 4-Queens Problem





Example: 4-Queens Problem





Forward Checking Issues

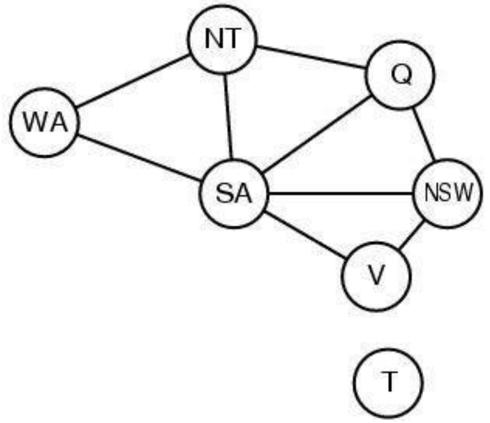
Solving CSPs (backtrack) with combination of heuristics plus forward checking is more efficient than either approach alone

But Forward Checking **does not see all inconsistencies.**

Consider our map coloring search, after we have assigned WA=red and Q=green

Forward checking

WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦
🟥	🟦	🟩	🟥🟦	🟥🟩🟦	🟦	🟥🟩🟦



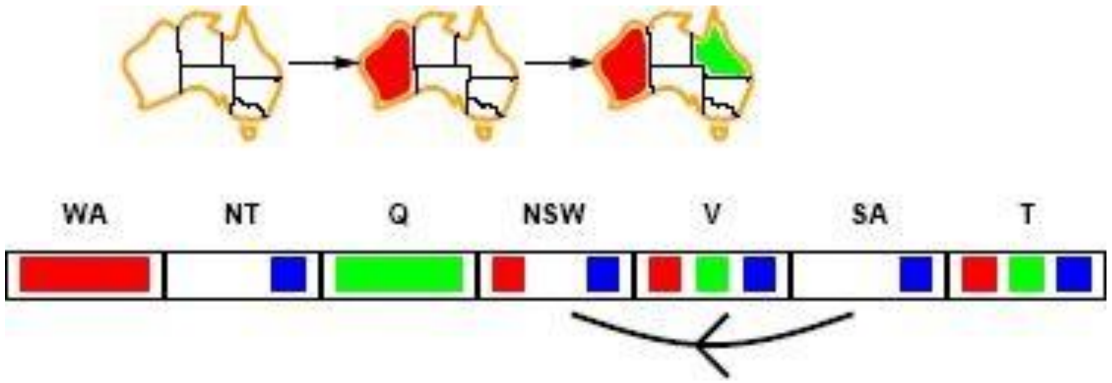
- WA=Red; *Q=green*
- Forward checking gives us the third row

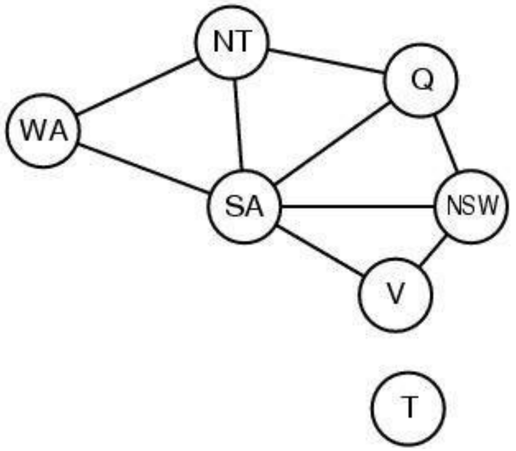
- At this point, we can see that this is inconsistent, since NT and SA are forced to be blue, yet they are adjacent.
- Forward checking doesn’t see this, and proceeds onward in the search from this state (as we saw earlier)

Constraint propagation

- **Forward checking (FC)** is in effect eliminating parts of the search space.
- Constraint propagation goes further than FC by repeatedly enforcing constraints locally
  - Needs to be faster than actually searching to be effective
- **Arc-consistency (AC)** is a systematic procedure for constraint propagation

Arc consistency



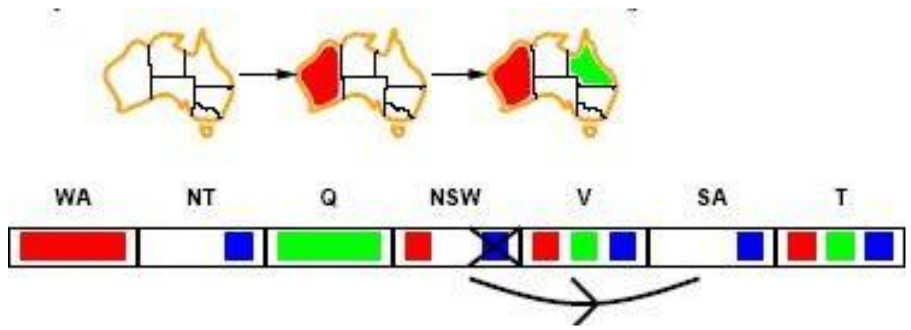


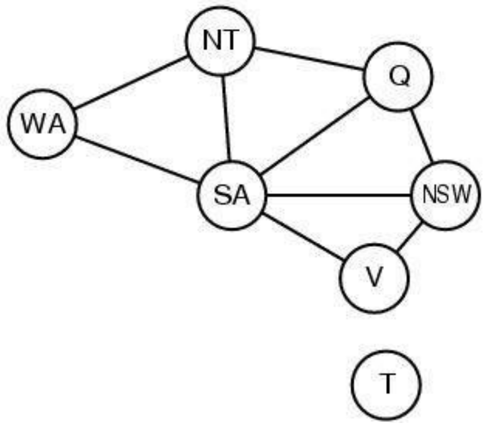
- *An Arc X* → *Y* is consistent if

  for *every* value *x* of *X* there is some value *y* consistent with *x*

- Consider state of search after WA and Q are assigned:
  - *SA* → *NSW* is consistent: if *SA=blue NSW could be =red*

Arc consistency





- *X* → *Y* is consistent if for *every* value *x* of *X* there is some value *y* consistent with *x*
- We will try to make the arc consistent by deleting x's for which there is no y (and then check to see if anything else has been affected – algorithm is in a few slides)

- *NSW → SA: if NSW=red SA could be =blue*

  But, if NSW=blue, there is no color for SA.

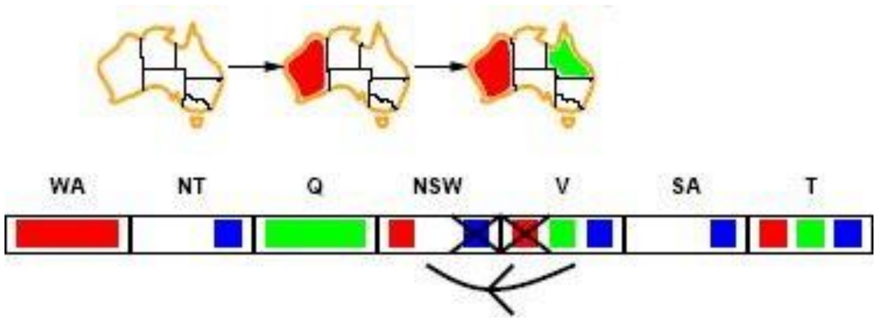
  So, remove blue from the domain of NSW

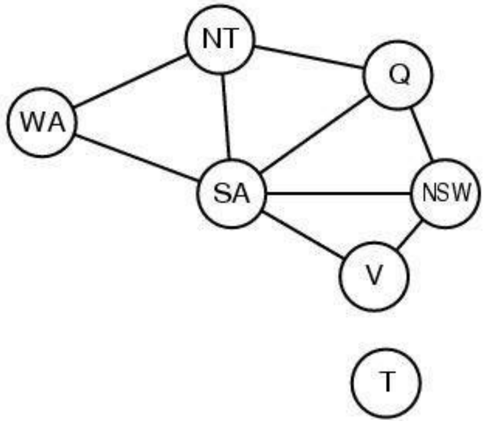
  Propagate the constraint: need to check Q □NSW SA □NSW V □NSW

  If we remove values from any of Q, SA, or V's domains, we will need to check THEIR neighbors

[continue process on next slide and board]

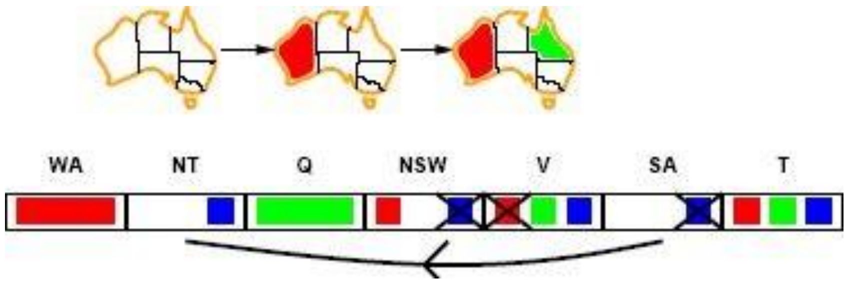
Arc consistency

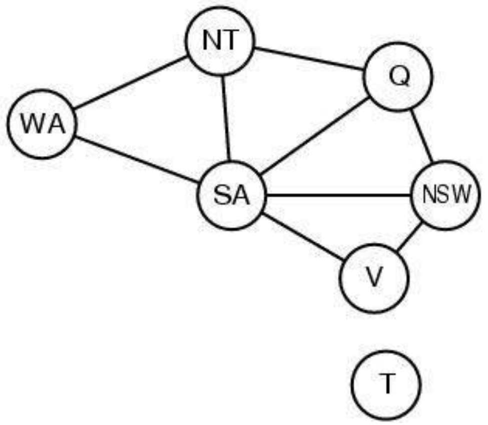




- After removing red from domain of V to make V □NSW arc consistent
- *SA □V, NSW □V check out; no changes*
- *Check the remaining arcs: most check out, until we check SA □NT, NT □SA. Whichever is checked first will result in failure.*

Arc consistency





- *SA* → *NT* is not consistent
  - and cannot be made consistent

- **Arc consistency detects failure earlier than FC**
- This process was all in one call to the INFERENCE function right after we assigned Q=green.
- Forward checking proceeded in the search, assigning a value to V.

Arc consistency checking

- **AC must be run until no inconsistency remains.**

- Trade-off
  - Requires some overhead to do, but generally more effective than direct search
  - In effect it can eliminate large (inconsistent) parts of the state space more effectively than search can

- Need a systematic method for arc-checking
  - If *X* loses a value, neighbors of *X* need to be rechecked:
    i.e. incoming arcs can become inconsistent again

    (outgoing arcs will stay consistent).

Arc consistency algorithm (AC-2)

```
function AC-2(csp) returns false if inconsistency found, else true, may reduce csp domains
    local variables: queue, a queue of arcs, initially all the arcs in csp
        /* initial queue must contain both ($X_i$, $X_j$) and ($X_j$, $X_i$) */
    while queue is not empty do
        ($X_i$, $X_j$) ← REMOVE-FIRST(queue)
        if REMOVE-INCONSISTENT-VALUES($X_i$, $X_j$) then
                if size of $D_i = 0$ then return false
                for each $X_k$ in NEIGHBORS[$X_i$] −{$X_j$} do
                        add ($X_k$, $X_i$) to queue if not already there
    return true

function REMOVE-INCONSISTENT-VALUES($X_i$, $X_j$) returns true iff we delete a
        value from the domain of $X_i$
    removed ← false
    for each x in DOMAIN[$X_i$] do
        if no value y in DOMAIN[$X_j$] allows (x,y) to satisfy the constraints
                between $X_i$ and $X_j$
        then delete x from DOMAIN[$X_i$]; removed ← true
    return removed
```

Complexity of AC-2

- A binary CSP has at most **n^2** arcs
- Each arc can be inserted in the queue **d** times (worst case)
  - (X, Y): only **d** values of **X** to delete
- Consistency of an arc can be checked in $O(d^2)$ time (d values of the first * d values of the second)
- Complexity is **$O(n^2 d^3)$**
- Although substantially more expensive than Forward Checking, Arc Consistency is usually worthwhile.

Trade-offs

- Running stronger consistency checks:
  - Takes more time
  - But will reduce branching factor and detect more inconsistent partial assignments

  - No “free lunch”
    - In worst case n-consistency takes exponential time

- Generally helpful to enforce 2-Consistency (Arc Consistency)

- Sometimes helpful to enforce 3-Consistency

- Higher levels may take more time to enforce than they save.

CSP Solvers

1. **AMPL** system, C++, C#, Java, MATLAB, Python, and R callable library
   https://ampl.com/
2. **GUROBI, C and C++ callable library**
   https://www.gurobi.com/documentation/9.0/refman/lp_format.html
3. **MATLAB**, the intlinprog function
   https://www.mathworks.com/help/optim/ug/intlinprog.html
4. **GAMS** language, C++, .NET, Java, and Python callable library
   https://www.gams.com/
5. **Z3**, C++ and Python callable library
   https://github.com/Z3Prover/z3
   https://theory.stanford.edu/ nikolaj/programmingz3.html
6. **Pulp**, Python callable library
   https://coin-or.github.io/pulp/
   http://benalexkeen.com/linear-programming-with-python-and-pulp-part-2 /
7. **p_solve**, C and C++ callable library
   http://lpsolve.sourceforge.net/5.5/