RUTGERS

THE STATE UNIVERSITY
OF NEW JERSEY

Advance Search

Constraint Satisfaction Problems

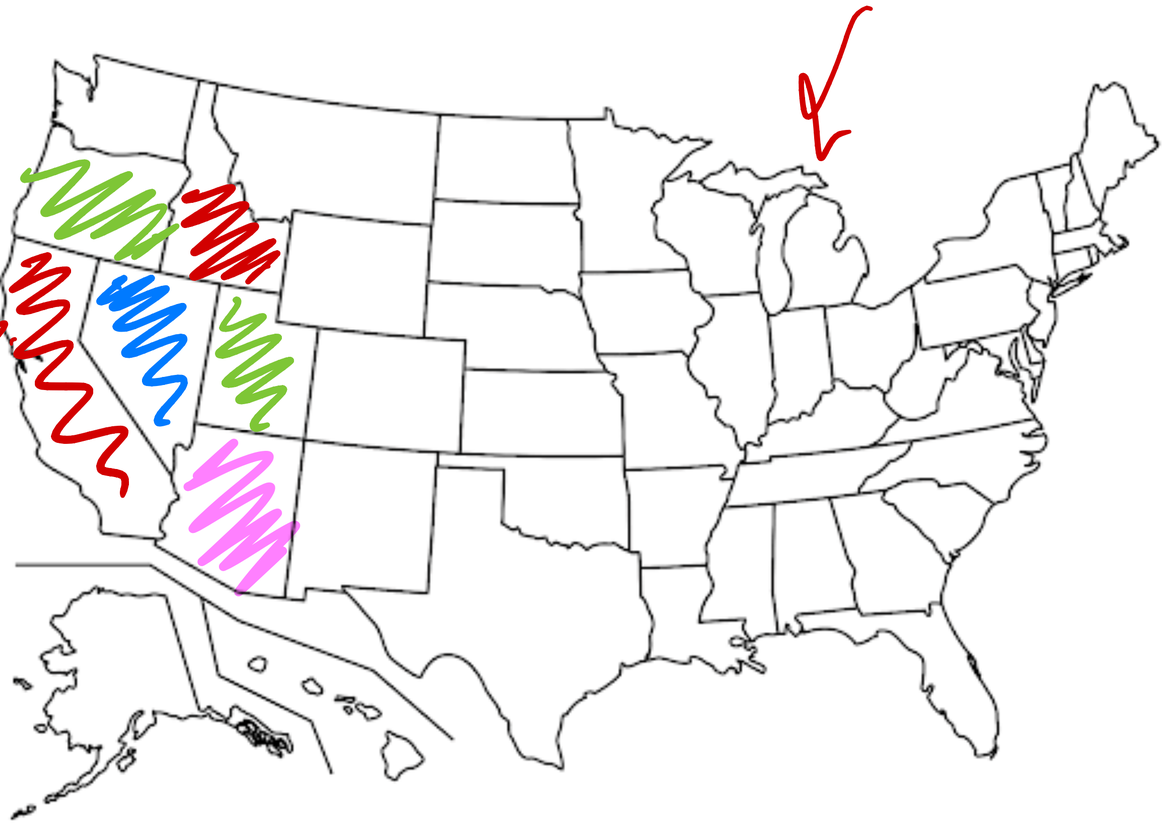
Edgar Granados

Coloring Map

As classic search

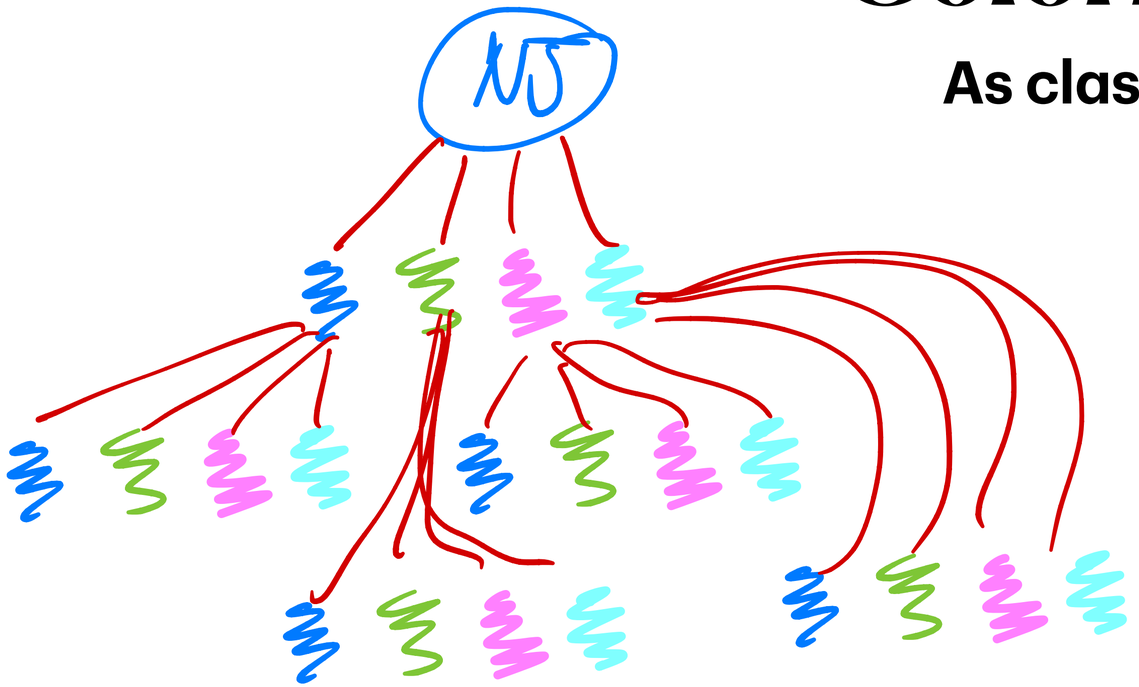
Problem

- **States:** An assignment of values to some or all
- **Initial State:** Empty Map
- **Goal State(s):** All states are colored different than its neighbors
- **Actions:** Color state
- **Transition model:** Ret.
- **Cost:** 1



Coloring Map

As classic search



Constraint Satisfaction Problems

Definition

- Problem where a state is defined as a set of constraints
- **X**: Set of variables $\{x_1, \ldots, x_n\}$
- **D**: Set of domains $\{D_1, \ldots, D_n\}$. $x_i \in D_i$
- **C**: Set of constraints

Constraint Satisfaction Problems

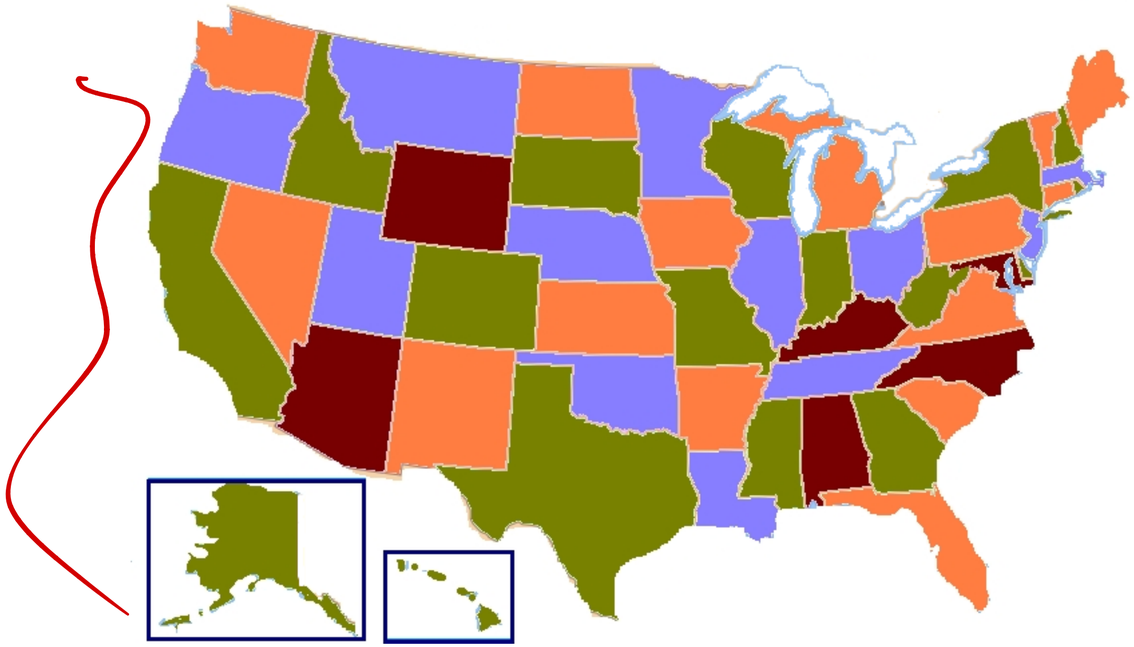
State Space

- **State**: assignment of values to some or all of the variables
- **Consistent** Assignment: assignment not violating any constraint
- **Partial Assignment**: Assignment to some variables
- **Complete Assignment**: Assignment to all variables
- **Solution**: Consistent complete assignment

CSP

Map Coloring

- **Four color theorem** (Appel and Haken, 1976)
  - Given any separation of a plane into contiguous regions, no more than four colors are required to color the regions of the map so that no two adjacent regions have the same color

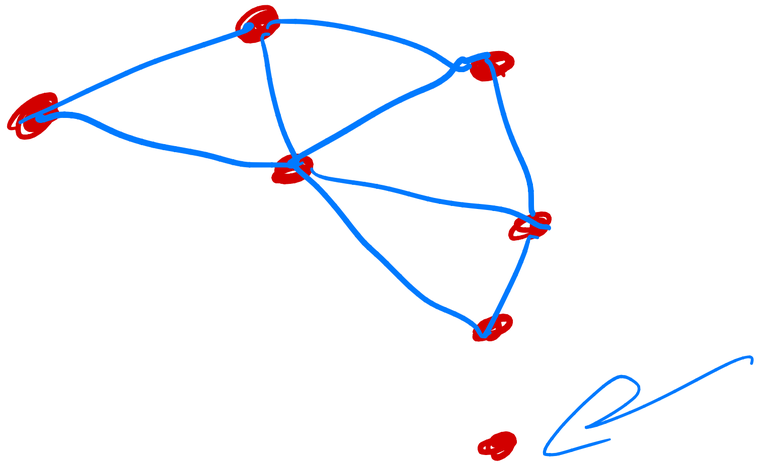


Map Coloring

Example

1 2 i

- **X** = {WA, NT, SA, Q, NSW, V, T}
- **D** => D_i = {red, green, blue}
- **C** = {SA ≠ WA, WA ≠ NT, SA ≠ NT, Q ≠ SA, SA ≠ NSW, SA ≠ V, WA ≠ NT, NT ≠ Q, Q ≠ NSW, NSW ≠ V}

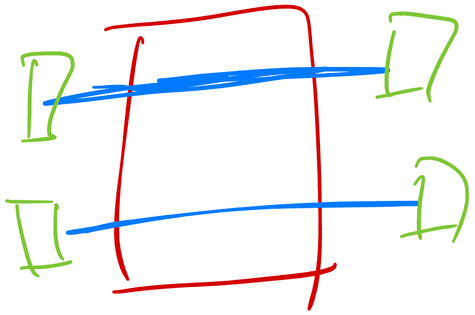


CSP

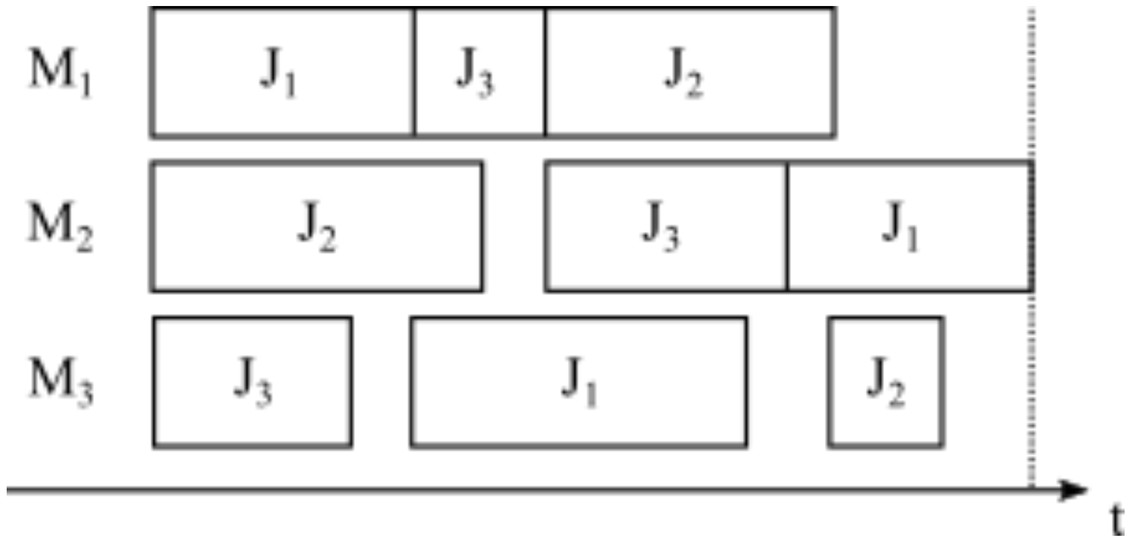
- Why formulate a problem as a CSP?
- When to formulate a problem as a CSP?

CSP

Job-Shop Scheduling



- Car assembly
  1. Install Axes - 10 mins each
  2. Affix wheels - 1 min each
  3. Tighten nuts - 2 mins each
  4. Affix hubcaps - 1 min each
  5. Inspect - 3 mins



CSP

Job-Shop Scheduling

- Car assembly
  1. Install Axes - 10 mins each
  2. Affix wheels - 1 min each
  3. Tighten nuts - 2 mins each
  4. Affix hubcaps - 1 min each
  5. Inspect - 3 mins

- (Each variable indicated the starting time of the corresponding task)
- **X** =
- **D** =
- **C** =

Job-Shop Scheduling

Disjunctive Constraints

- Car assembly
  1. Install Axes - 10 mins each
  2. Affix wheels - 1 min each
  3. Tighten nuts - 2 mins each
  4. Affix hubcaps - 1 min each
  5. Inspect - 3 mins

$i = \{F, B\}$ $j = \{R, L\}$

- Suppose there are four workers to install wheels, but they have to share one tool that helps put the axle in place.
- **C** =

$$A_i + 10 \leq W_{ij}$$

$$W_{ij} + 1 \leq N_{ij}$$

$$N_{ij} + 2 \leq C_{ij}$$

$$C_{ij} + 1 \leq I$$

New C_k

$$A_F + 10 \leq A_B$$

OR

$$A_B + 10 \leq A_F$$

Job-Shop Scheduling

Linear Programming

- Car assembly
  1. Install Axes - 10 mins each
  2. Affix wheels - 1 min each
  3. Tighten nuts - 2 mins each
  4. Affix hubcaps - 1 min each
  5. Inspect - 3 mins

$[0, 27]$

- How to model this using linear programming?

$\min$

$s.t.$

$Inspect$

$\forall X_k \in X - \{Inspect\} : X_i + T_i \leq Inspect$

$A_i + 10 \leq W_{ij}$

$W_{ij} + 1 \leq N_{ij}$

$N_{ij} + 2 \leq C_{ij}$

$C_{ij} + 1 \leq I$

$i = \{F, B\} \quad j = \{R, L\}$

CSP

Example: Crypt-arithmetic puzzle

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

- **X:** $\{T, W, O, F, U, R\}$
- **D:** $D_i = \{0,1,2,3,4,5,6,7,8,9\}$
- **C:** alldiff(T, W, O, F, U, R)

CSP: Crypt-arithmetic puzzle

TWO + TWO = FOUR

Linear programming?

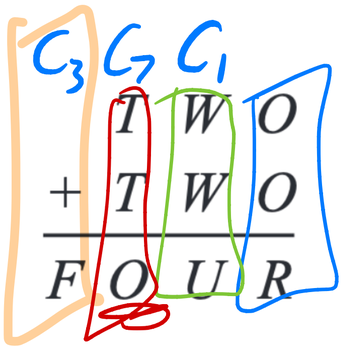
Min

S.t.,

$0 \le x_i \le 9$

$x_i \neq x_j$

$$\begin{array}{r} 0\ \boxed{1} \\ T\ W\ O \\ +\ T\ W\ O \\ \hline F\ O\ U\ R \end{array}$$



CSP: Crypt-arithmetic puzzle

Hypergraph

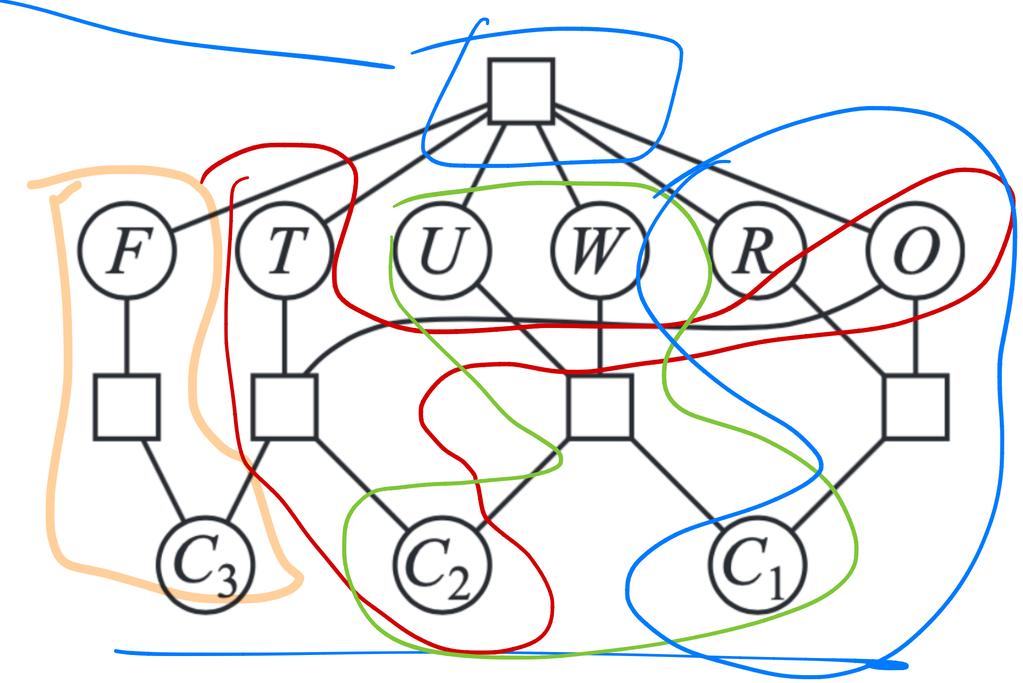
- Graph showing n-ary constraints
  - All different constraint
  - Column constraints

$$O + O = R + 10 \cdot C_1$$

$$C_1 + W + W = U + 10 C_2$$

$$C_2 + T + T = O + 10 C_3$$

$$C_3 = F$$



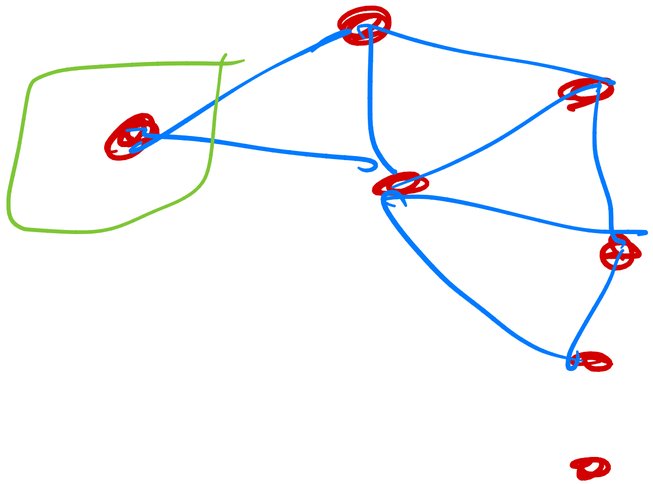
Constraint Propagation

- Shrink the domain of each variable based on the constraints
- Faster search
- Force consistency:
  - Node consistency
  - Arc Consistency
  - Path Consistency
  - K-consistency

Constraint Propagation

Node Consistency

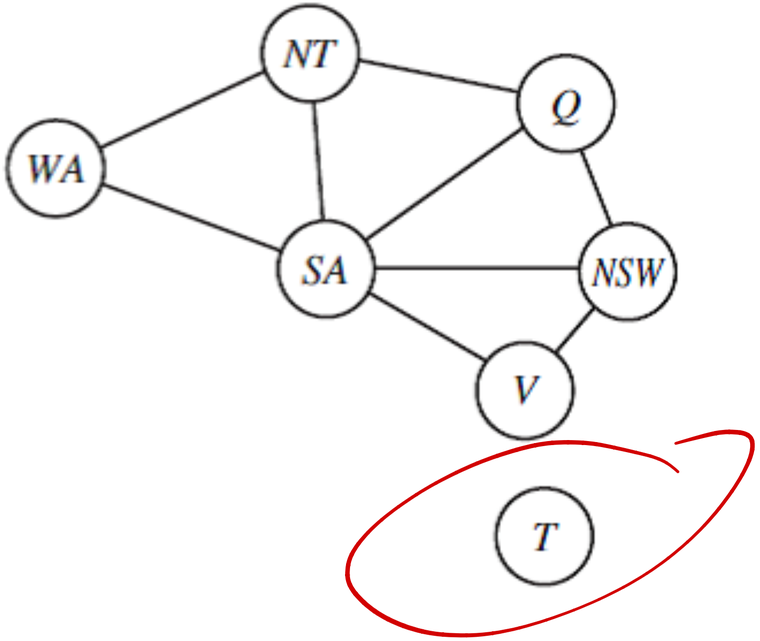
- A variable is node consistent if all values in the variables domain satisfy the variables unary constraints
- A network is node-consistent if every variable in the network is node consistent



Constraint Propagation

Arc Consistency

- A variable is arc consistent if all values in the variables domain satisfy the variables binary constraints
- X_i is arc-consistent with respect to X_j if $\forall d \in D_i, \exists d' \in D_j$ that satisfies (X_i, X_j)



Arc Consistency

Example

- **X:** X_1, X_2
- **D:** $D_i = \{0,1,2,3,4,5,6,7,8,9\}$
- **C:** $X_1 = \sqrt{X_2}$

$D_1 = \{0, 1, \ldots, 9\}$ { 10

$D_2 = \{0, 1, \ldots, 9\}$ { 10

Make X_1 arc consistent with X_2 $D_1 = \{0, 1, 2, 3\}$ { 4

Make X_2 arc consistent with X_1 $D_2 = \{0, 1, 4, 9\}$ { 4

Arc Consistency

Example

function AC-3(*csp*) **returns** false if an inconsistency is found and true otherwise
  inputs: *csp*, a binary CSP with components (X, D, C)
  local variables: *queue*, a queue of arcs, initially all the arcs in *csp*

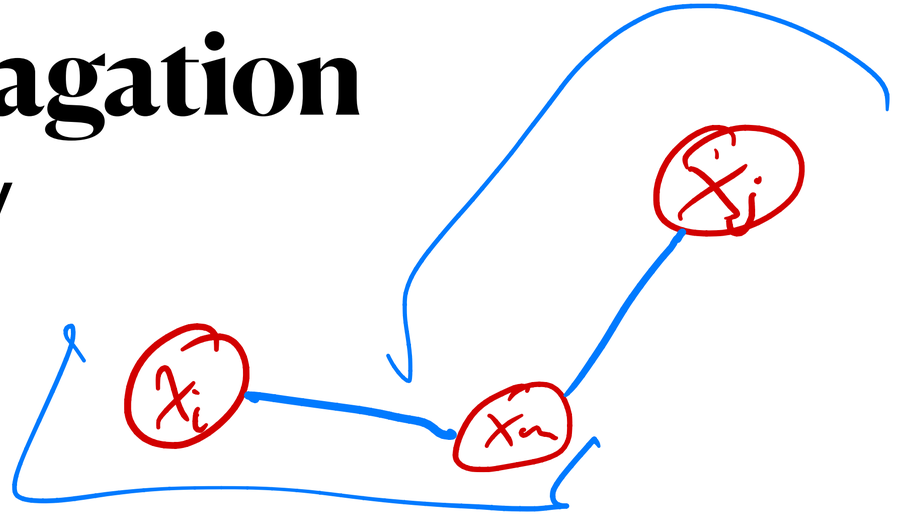
  while *queue* is not empty **do**
    $(X_i, X_j) \leftarrow$ REMOVE-FIRST(*queue*)
    if REVISE(*csp*, X_i, X_j) **then**
      if size of $D_i = 0$ **then return** *false*
      for each X_k **in** X_i.NEIGHBORS - $\{X_j\}$ **do**
        add (X_k, X_i) to *queue*
  return *true*

---

function REVISE(*csp*, X_i, X_j) **returns** true iff we revise the domain of X_i
  revised $\leftarrow$ *false*
  for each x **in** D_i **do**
    if no value y in D_j allows (x,y) to satisfy the constraint between X_i and X_j **then**
      delete x from D_i
      revised $\leftarrow$ *true*
  return *revised*

Constraint Propagation

Path Consistency



- (X_i, X_j) is path consistent with respect to X_m if:
  - [illegible]
  - An assignment exists such that (X_i, X_m) is arc consistent
  - An assignment exists such that (X_m, X_j) is arc consistent

Constraint Propagation

Path Consistency

- (X_i, X_j) is path consistent with respect to X_m if:
  - (X_i, X_j) is arc consistent
  - An assignment exists such that (X_i, X_m) is arc consistent
  - An assignment exists such that (X_m, X_j) is arc consistent

Constraint Propagation

K-Consistency

- A CSP is k-consistent if, for any set of k-1 variables and for any consistent assignment to this variables, a consistent value can always be assigned to any kth variable
  - 1-consistent → Node
  - 2-consistent → Arc
  - 3-consistent → Path

Constraint Propagation

K-Consistency

- A CSP is k-consistent if, for any set of k-1 variables and for any consistent assignment to this variables, a consistent value can always be assigned to any kth variable
- A CSP is strongly k-consistent if it is k-consistent and is also (k-1)-consistent,(k-1)-consistent, (...), 1-consistent
  - If a CSP has n variables, and it is strongly n-consistent, then a solution can be easily found by setting any variable to any value from its reduced domain

Example

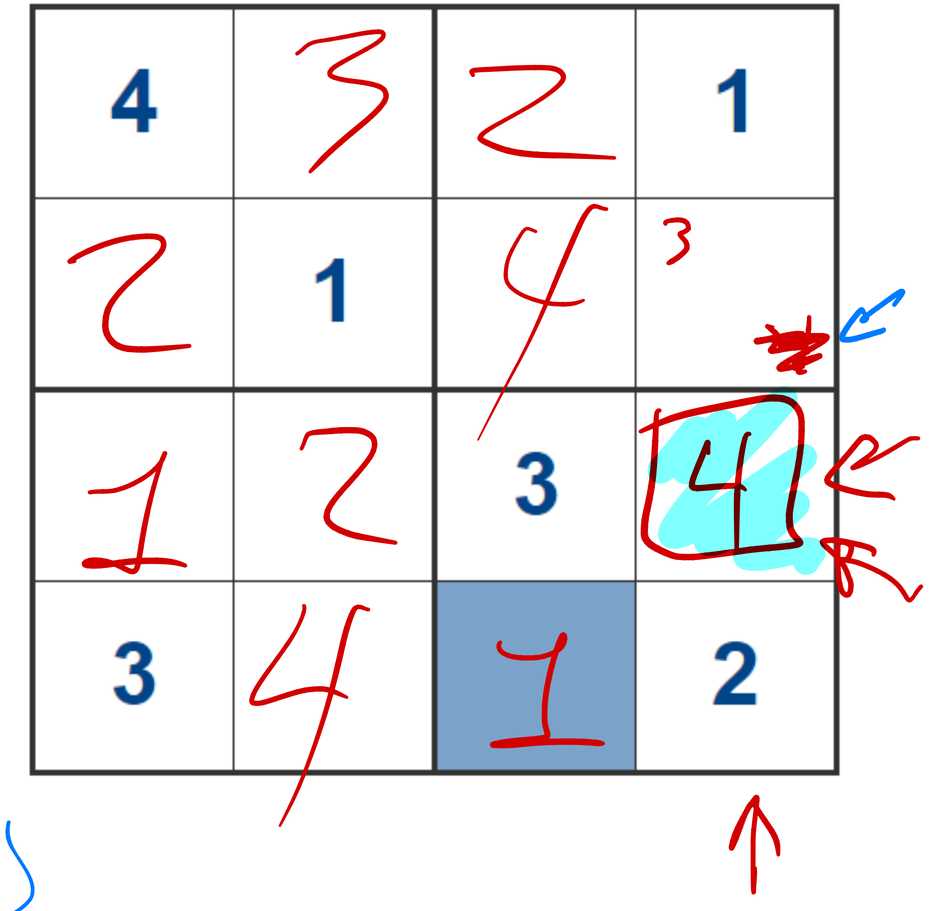
Sudoku

- **X:** $\{x_{ij}\}\ j,i \in \{1,2,3,4\}$
- **D:** $D_{ij} = \{1,2,3,4\}$
- **C:** $x_{i1} \neq x_{i2} \neq x_{i3} \neq x_{i4}$ $\quad i \in \{1,2,3,4\}$

$x_{1j} \neq x_{2j} \neq x_{3j} \neq x_{4j}$ $\quad j \in \{1,2,3,4\}$

$(x_{11} \neq x_{12} \neq x_{21} \neq x_{22}) \cap$

$(x_{13} \neq x_{14} \neq x_{23} \neq x_{24}) \cap (\ldots)$



Constraint Propagation

Backtracking Search

- Inference is not enough to solve CSP
  - **X:** X_1, X_2
  - **D:** $D_i = \{0,1,2,3,4,5,6,7,8,9\}$
  - **C:** $X_1 = \sqrt{X_2}$

$X_1 = f(X_2)$

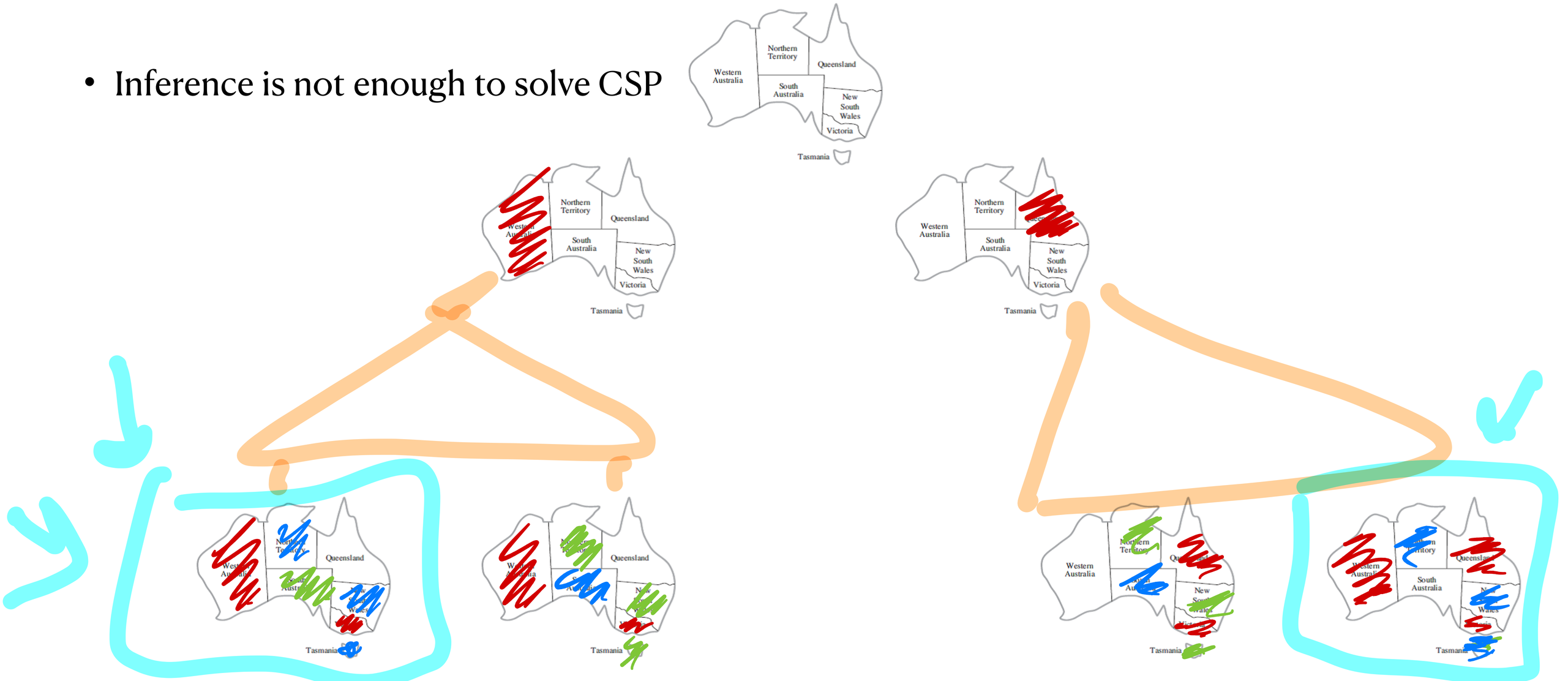
$D_1 = \{0, 1, 2, 3\}$

$D_2 = \{0, 1, 4, 9\}$

Constraint Propagation

Backtracking Search

- Inference is not enough to solve CSP



Constraint Propagation

Backtracking Search

- Inference is not enough to solve CSP
- **Commutativity**
  - A problem is commutative if the order of application of the set of actions does not matter

Constraint Propagation

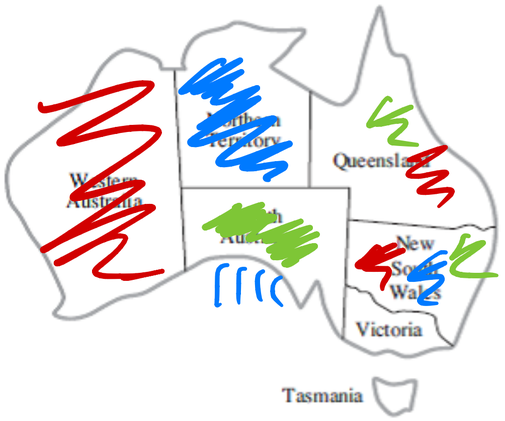
Backtracking Search

- Inference is not enough to solve CSP
- **Commutativity**
  - A problem is commutative if the order of application of the set of actions does not matter
- Use *domain-independent* heuristics

Backtracking Search

Variable and Value Ordering

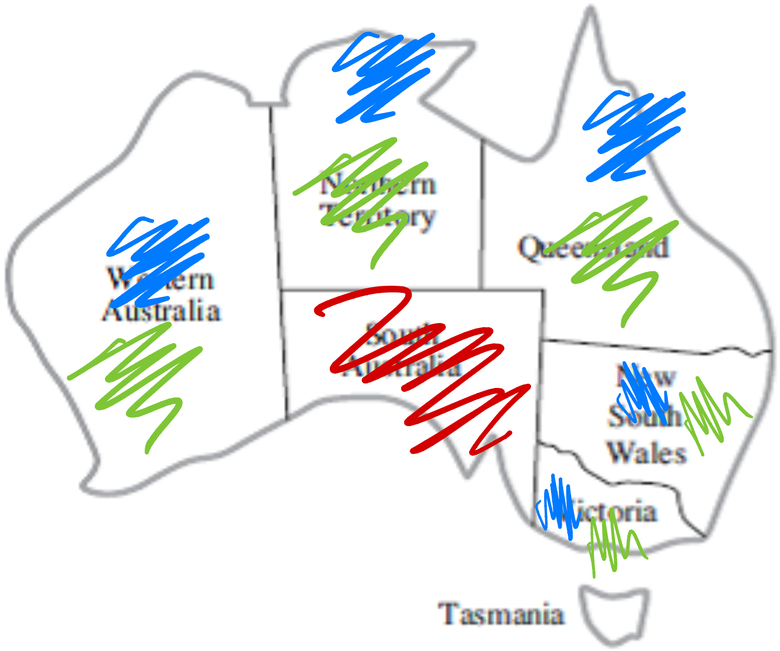
- Minimum remaining-values heuristic

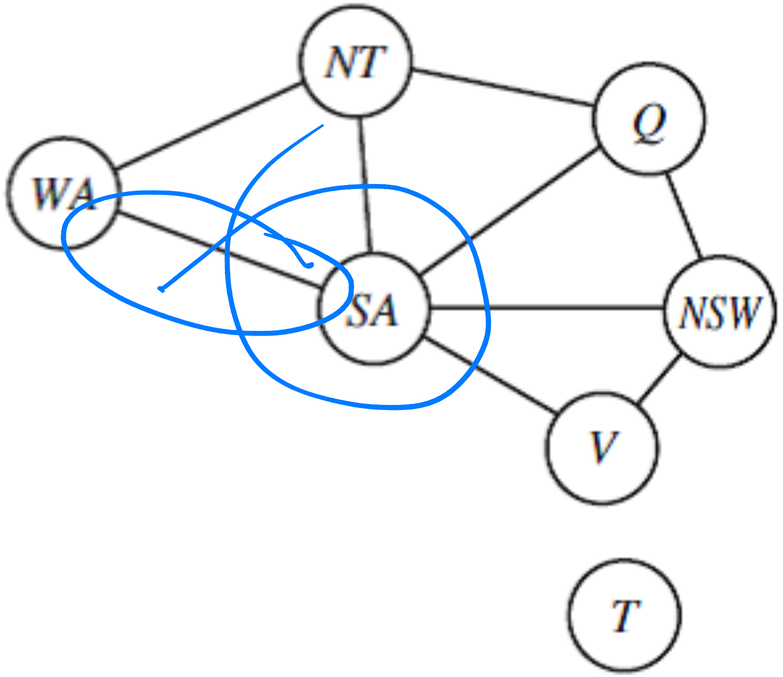


Backtracking Search

Variable and Value Ordering

- Degree heuristic: Select the variable that is involved in the largest number of constraints on other unassigned variables

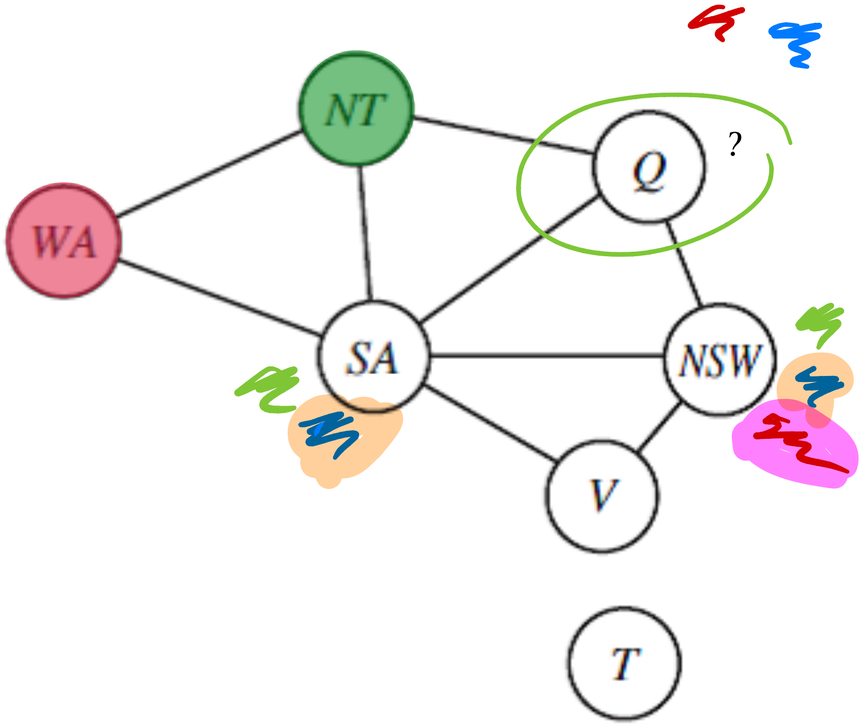




Backtracking Search

Variable and Value Ordering

- Least-constraining-value: Select the variable that rules out the fewest choices for the neighboring variables in the constraint graph



Backtracking Search

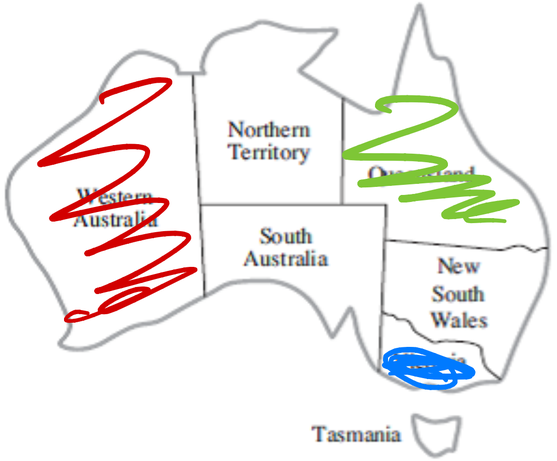
```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
    return BACKTRACK({ }, csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(csp)
    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, var, value)
            if inferences ≠ failure then
                add inferences to assignment
                result ← BACKTRACK(assignment, csp)
                if result ≠ failure then
                    return result
        remove {var = value} and inferences from assignment
    return failure
```

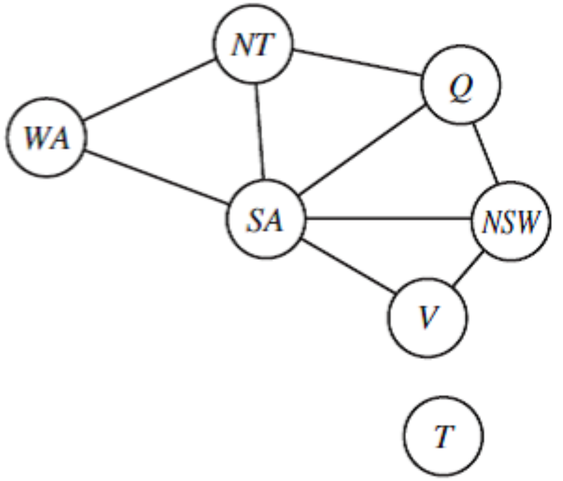
Backtracking Search

Intelligent Backtracking

- Forward Search

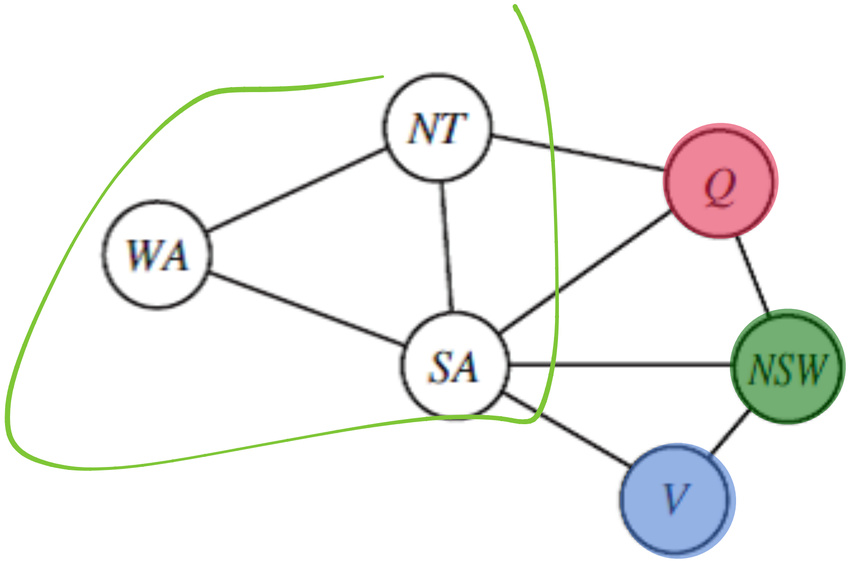
	WA	NT	Q	NSW	V	SA	T
Initial domains							
After WA=red							
After Q=green							
After V=blue							





Backtracking Search

Interleaving search and inference



What if T goes next?

	WA	NT	Q	NSW	V	SA	T
Initial domains							
After *WA=red*							
After *Q=green*							
After *V=blue*							

CSP

More (possible) Improvements

- Local Search
- Structure of the problem
  - Connected components
  - Topological sort