Problem Solving Using Search

IT426: Artificial Intelligence

Information Technology Department

Constraint Satisfaction Problems

Agents

- Simple reflex agents

  Select actions based on the *current* percept

- Model-based reflex agents

  Knowledge about "how the world works"

- **Goal-based agents (Problem Solving Agents)**

  "What will happen if I do such-and-such?" and "Will that make me happy?"

- Utility-based agents

  Exactly how happy

- Learning agents

Atomic vs. Factored States

- Atomic state: no internal structure.
  - E.g.: In(Arad)
  - Standard search problems

- Factored state: has a set of variables, each of which has a value
  - E.g.:{C1=Red, C2=Green, C3=Blue}
- The problem is solved if each variable has a value that satisfies all the constraints.
  - Constraint Satisfaction Problems
- This representation allows defining general purpose heuristics rather than problem-specific ones.

Constraint Satisfaction Problems

Main idea: eliminate large portions of the search space all at once by identifying variables/value combinations that violate the constraints.

https://cdn.shopify.com/s/files/1/2005/6615/products/product-image-833545284.jpg?v=1544664852

Definitions

- A CSP is defined by 3 components:
  - X: a set of variables, { X1,…,Xn} .
  - D: a set of domains {D1,…,Dn}, one for each variable.
  - C: a set of constraints C1, C2,…,Cn where each constraint Ci involves some subset of the variables and specifies the allowable combinations of values for that subset.

- **A state in a CSP:** is defined by an assignment of values to some or all the variables.

$$\{X_i = v_i, X_j = v_j, \ldots\}$$

- **Consistent assignment:** the one that does not violate any constraint (also called legal assignment).

- **Complete assignment:** the one in which every variable is mentioned.

Definitions

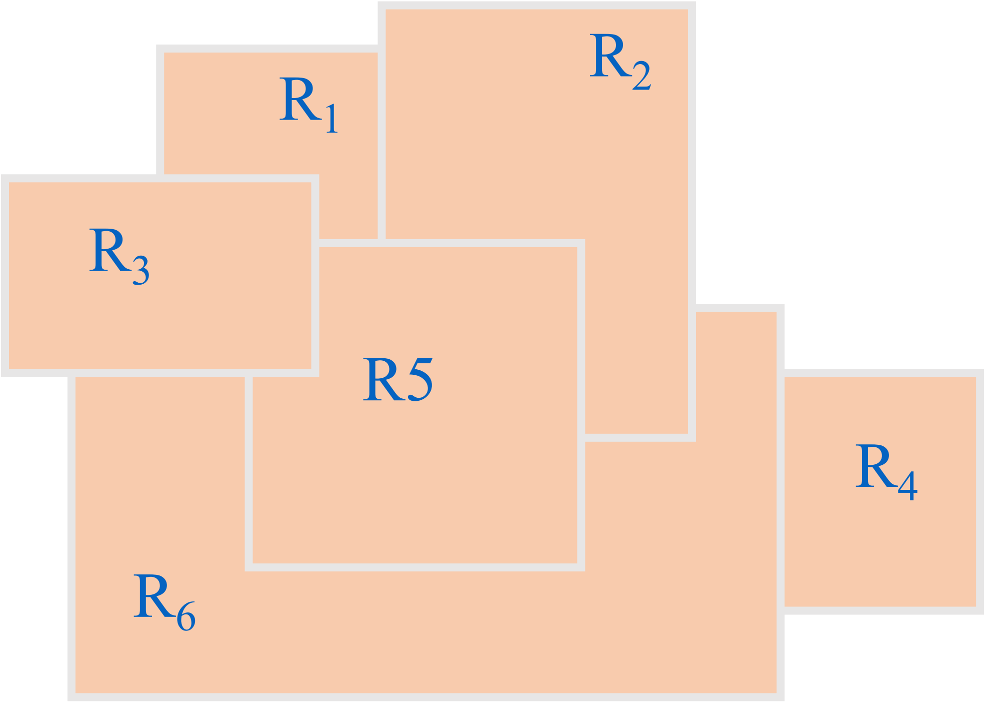
Solution in CSP: It is a complete assignment that satisfies all the constraints.

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

- **Constraint graph:** a CSP can be visualized by a constraint graph where nodes correspond to variables and arc to constraints.

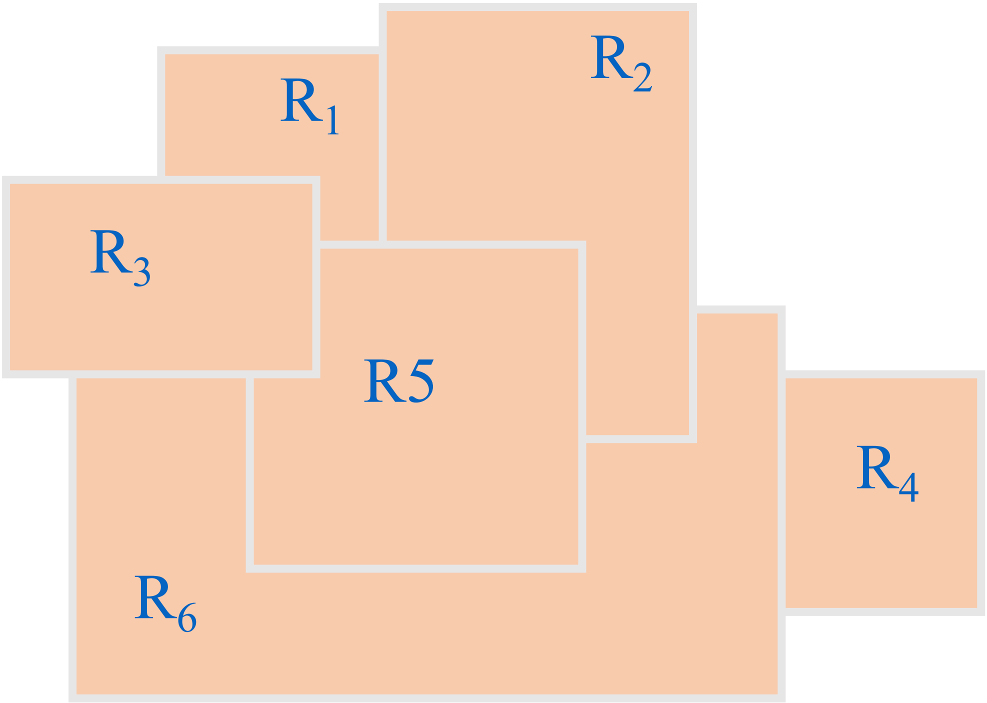
Map Coloring Example

Color a map so that **no** adjacent regions have same color using three colors.



Problem Definition

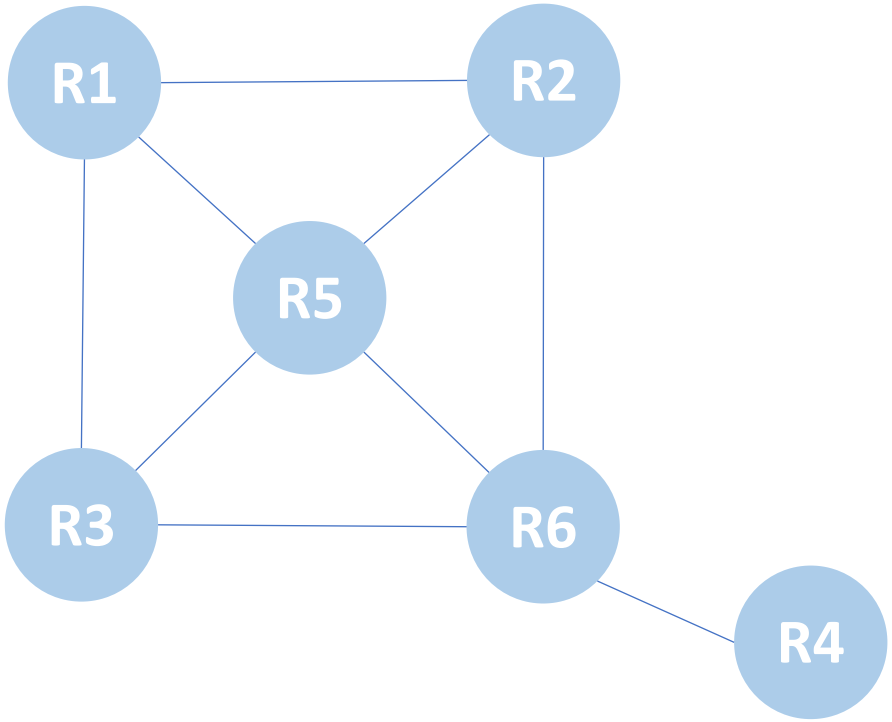
- **Variables:** Regions Ri, i=1 to i=6
- **Domains**: {Red, Blue, Green}
- **Constraints**: R1≠R2, R1≠R3, R1≠R5, R5≠R6, etc

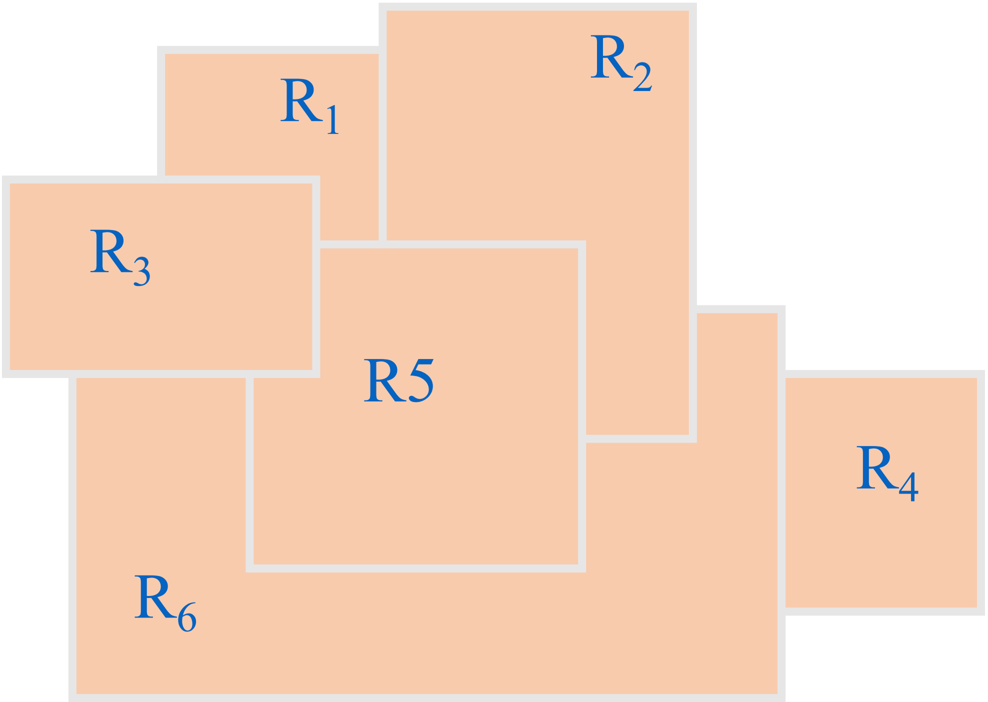


Constraint Graph

Draw a constraint graph for the color map problem.

Remember: nodes correspond to variables and arc to constraints.





Real world CSPs

- Assignment problems: e.g. who teaches what class?
- Timetabling problems: e.g. which class is offered, when and where?
- Transportation scheduling.
- Hardware configuration.
- Planning problems
- Etc …

CSP Formulation

- **Incremental formulation**

Involves operators that augment the state description, starting with an empty state; then progress to the next state by adding an assignment to a variable.

Consistent and legal always.

- **Complete formulation**

Every state is a complete assignment that might or might not satisfy the constraints.

Incremental Formulation

- **Initial state**: empty assignment {}, in which all variables are unassigned.
- **Transition model**: a value can be assigned to any unassigned variable provided that it does not conflict with previously assigned variables.
- **Goal test**: the current assignment is complete.
- **Path cost**: a constant cost for every step.

Depth is number of variables

Depth algorithm is suitable here

Questions:

- What is the depth of the search tree in this case? Which strategy is suitable?
- In a complete formulation, describe the following: initial state, transition model, goal test and path cost.

CSPs Varieties

- **Discrete variables**
  - with finite domains
    - e.g. Map coloring
    - Boolean CSPs, where variables can be either true of false.
  - with infinite domains
    - e.g. job scheduling when a deadline is not defined

  If $\boldsymbol{d}$ is the maximum domain size for any variable, and $\boldsymbol{n}$ is the number of variables, then the number of possible complete assignments is $\boldsymbol{d^n}$.

- **Continuous variables**
  - common in the real world; Continuous is easier for cap. Neural networks applications all use differentiation logic.
  - e.g. Hubble Space Telescope requires precise timing of observations;

Constraints Varieties

Unary constraint: involves a single variable.

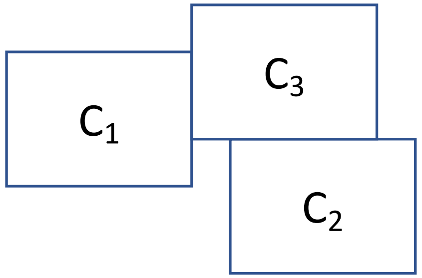
- e.g. C1 ≠ green

- **Binary constraint**: involves pairs of variables.
  - e.g. C1 ≠ C3
- **High order constraints**: involves 3 or more variables.
  - Value of Y is between X and Z, with the ternary constraint Between(X, Y,Z).
- **Global constraint**: involves an arbitrary number of variables but not necessarily all variables. (frequent in real world)

  - e.g. Alldiff constraint: all variables involved must have distinct values. (in Sudoku, all variables in a row or column must satisfy an Alldiff constraint)
- **Preferences (soft constraints):** hard constraints you need to satisfy. soft is just a preference.
  - e.g. red is better than blue
  - often represented by a cost for each variable assignment → constrained optimization problems

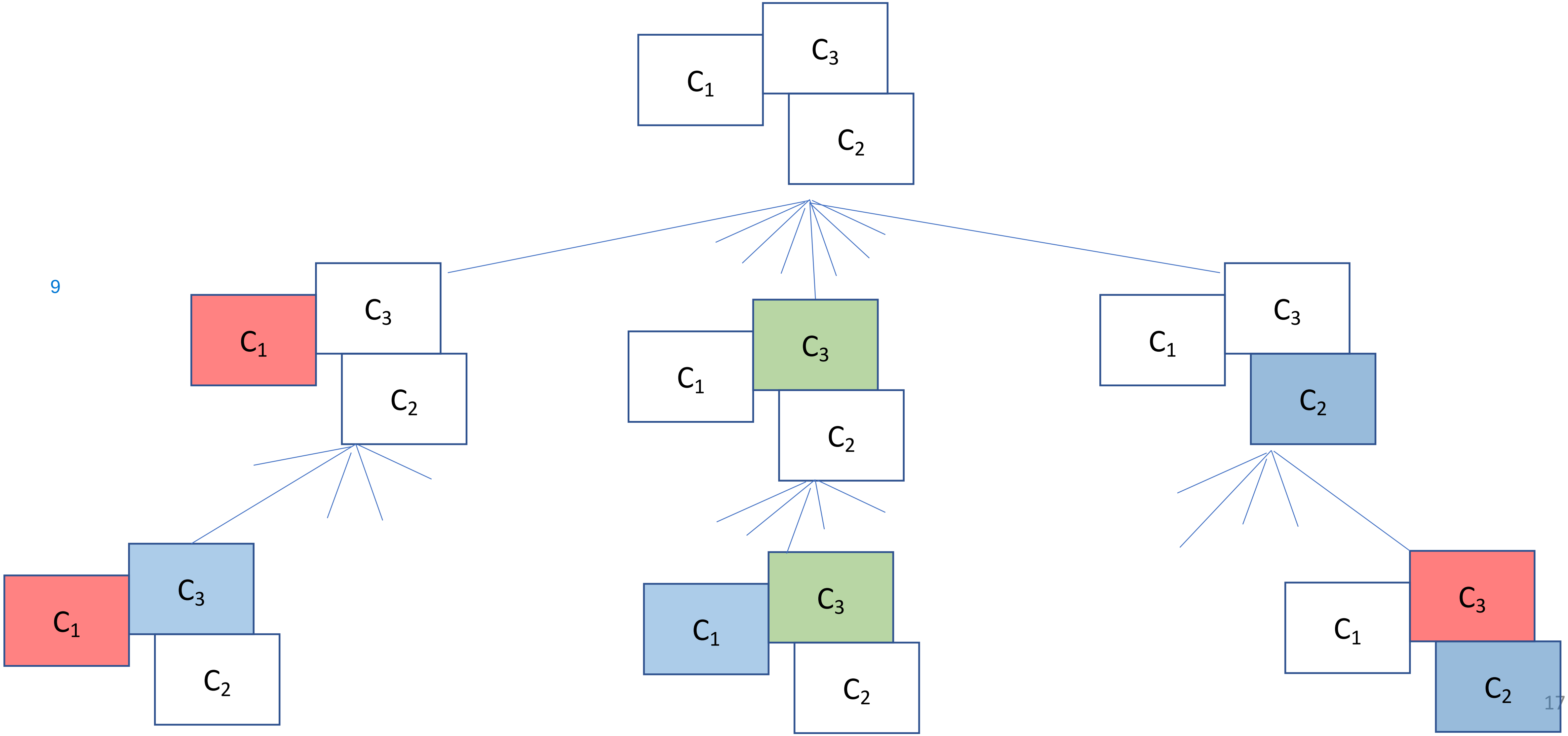
Map Coloring Problem

- Let's consider Map coloring problem with 3 regions (C_1, C_2, C_3) and 3 colors (RED, BLUE, GREEN).
- Initial state:



State Space In Incremental CSP

9

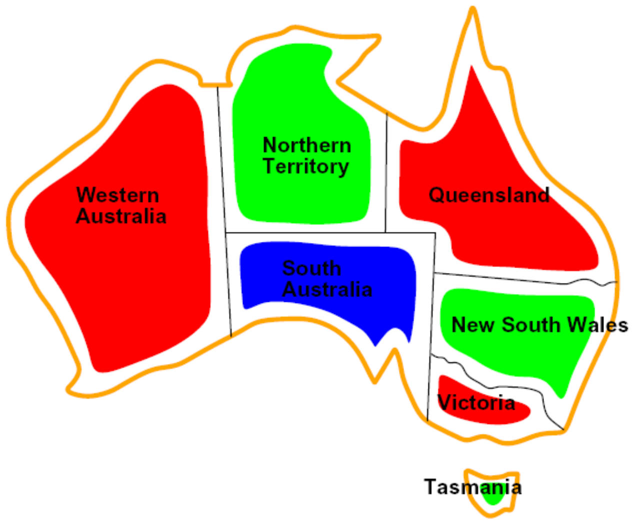


State Space Properties (Incremental Formulation)

- Maximum depth is n (number of variables).
- The depth of the solution is n.
- Branching factor at the top is nd (d: size of the domain).
- Branching factor at the next level is (n-1)d and the number of nodes generated in this level is (n-1)d * nd. Same thing for n next levels.
- Number of leaves is $n!d^n$ even though there are only d^n possible complete assignments.
- Suitable search technique is <u>DFS</u>.

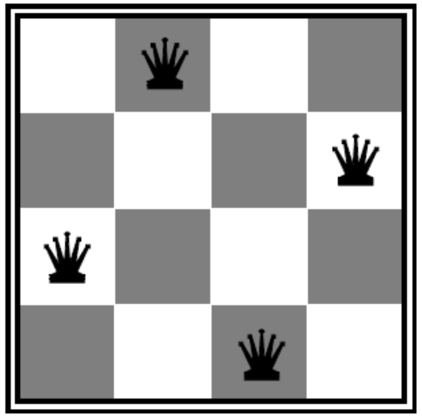
Formulation Example: Map Coloring

- Variables: WA, NT, Q, NSW, V, SA, T
- Domains: red, green, blue
- Constraints: adjacent regions must have different colors
  - WA≠ NT, WA ≠ SA, ...
- Solutions are assignments satisfying all constraints, e.g.: {WA=red, NT=green, Q=red, NSW=green, V=red, SA=blue, T=green}



Formulation Example: N-Queens

- Formulation 1:
  - Variables: X_{ij}
  - Domains: {0,1}
  - Constraints:

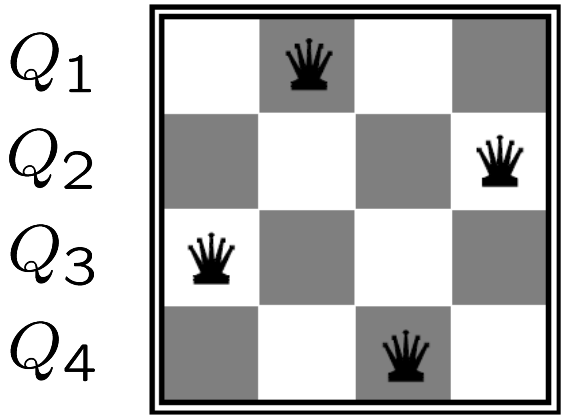


$$\forall i,j,k \;\; (X_{ij}, X_{ik}) \in \{(0,0),(0,1),(1,0)\}$$
$$\forall i,j,k \;\; (X_{ij}, X_{kj}) \in \{(0,0),(0,1),(1,0)\}$$
$$\forall i,j,k \;\; (X_{ij}, X_{i+k,j+k}) \in \{(0,0),(0,1),(1,0)\}$$
$$\forall i,j,k \;\; (X_{ij}, X_{i+k,j-k}) \in \{(0,0),(0,1),(1,0)\}$$

$$\sum_{i,j} X_{ij} = N$$

Formulation Example: N-Queens

- Think of another formulation given that:
  - Variables: Q_k
  - Domains?
  - Constraints?



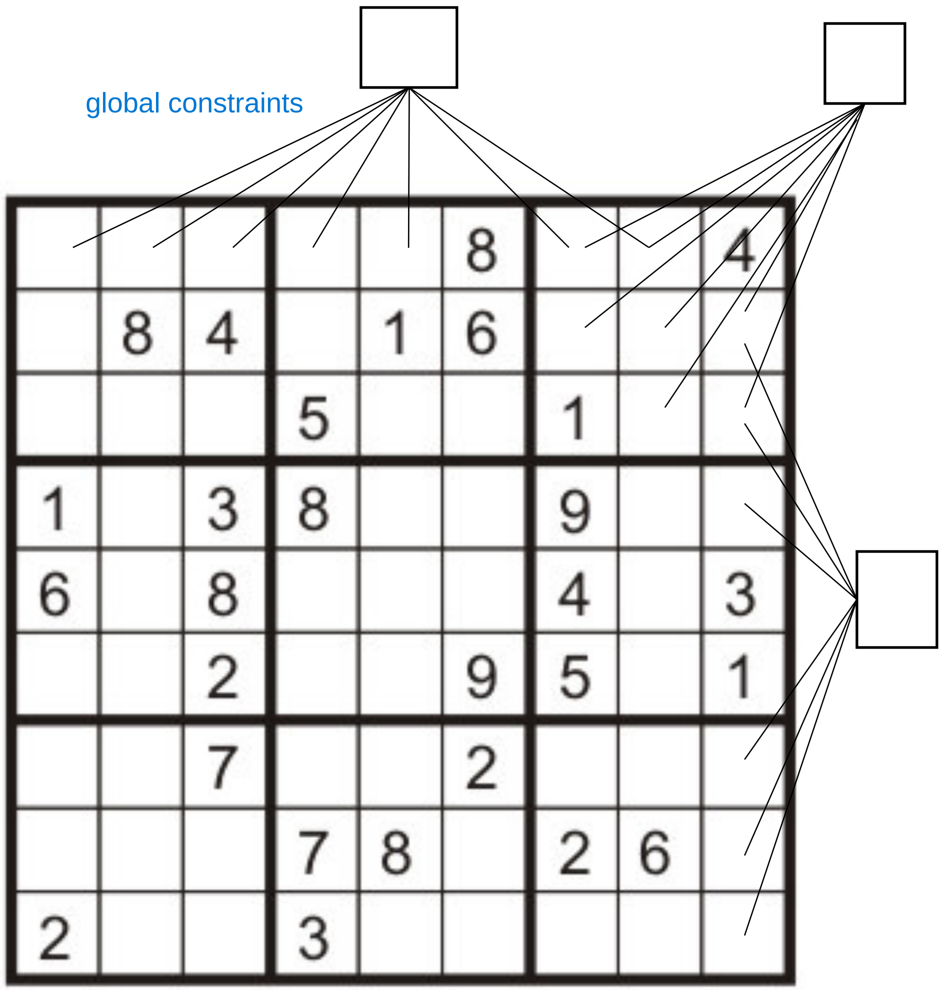
Formulation Example: Sudoku

- Variables:
  - Each (open) square
- Domains:
  - {1,2,…,9}
- Constraints:

  9-way alldiff for each row,

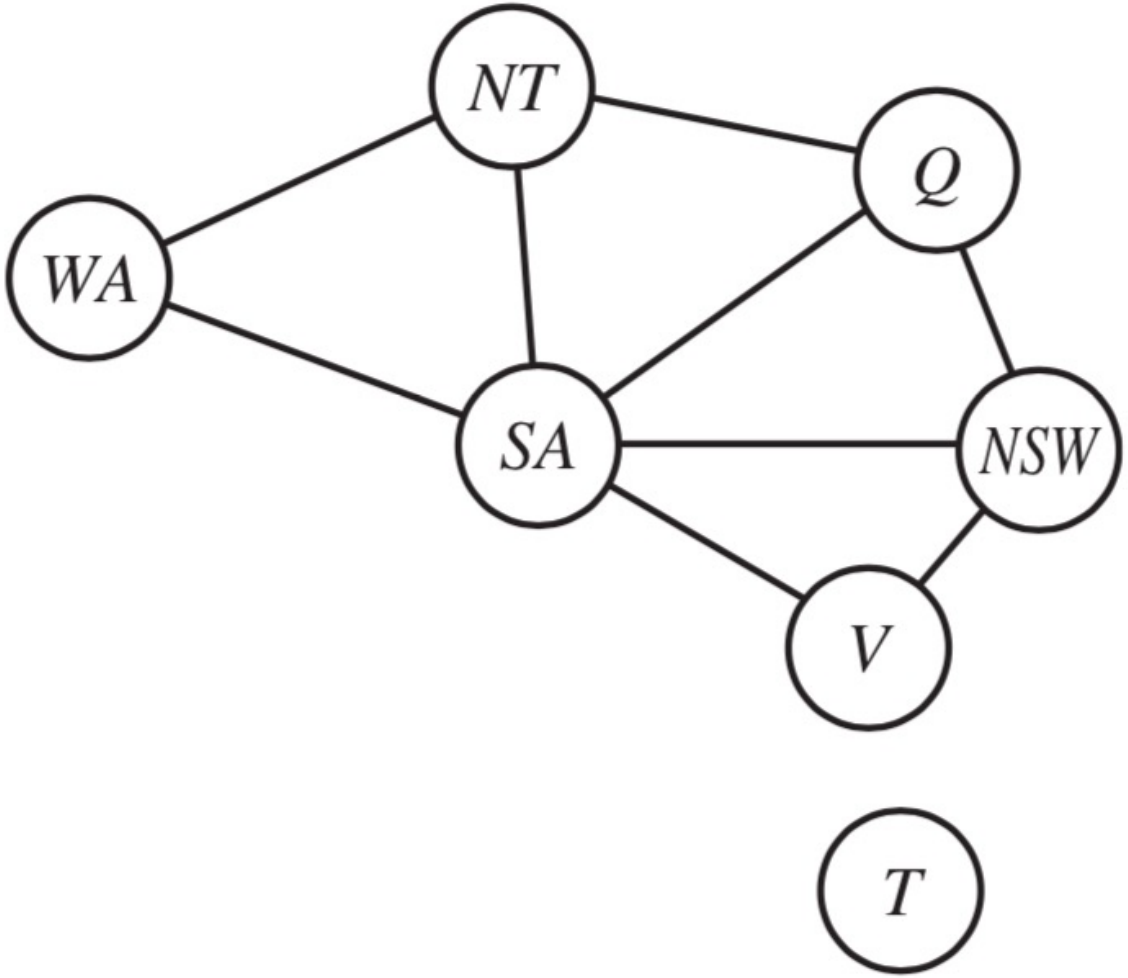
  9-way alldiff for each col and

  9-way alldiff for each region



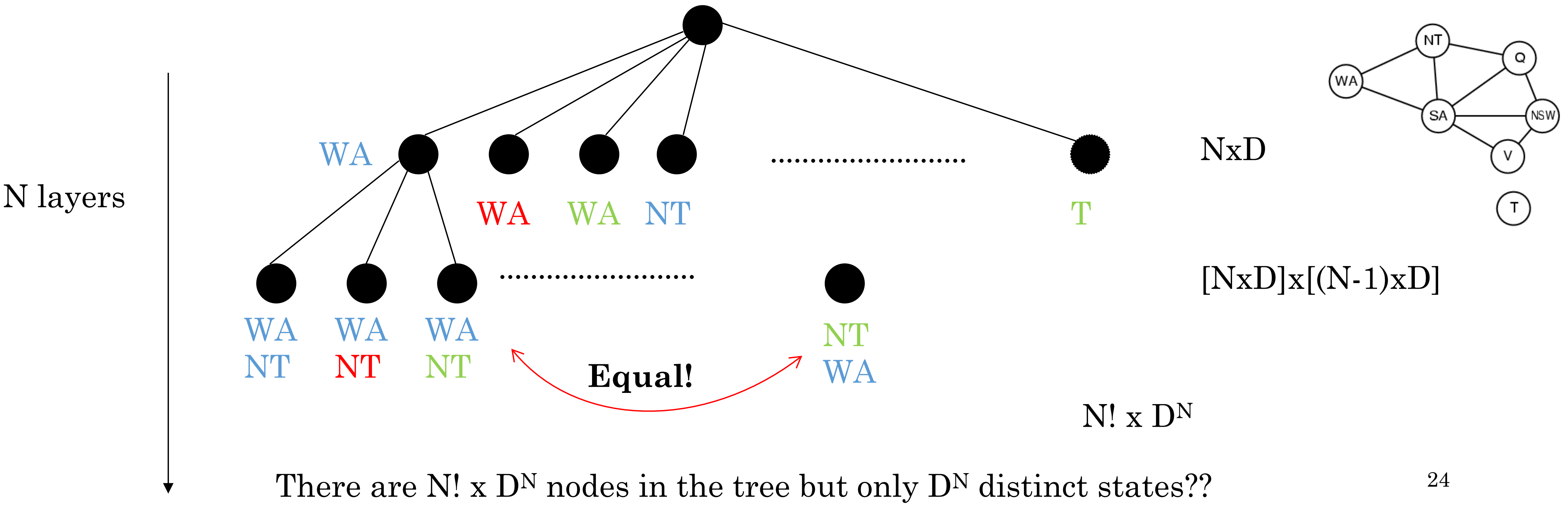
Australia Map

constraint graph



Standard Search Formulation

- Initial state: none of the variables has a value (color).
- Successor state: one of the variables without a value will get some value.
- Goal: all variables have a value and none of the constraints is violated.



There are $N! \times D^N$ nodes in the tree but only D^N distinct states??

Constraint Satisfaction Problems

This can be improved dramatically by noting the following:

- The formulation does not take into account one property of CSPs → Commutativity. In CSP the order of assignment is irrelevant, so many paths are equivalent; the order of application of any given set of actions has no effect on the outcome [R1 = red then R2 = green] is the same as [R2= green then R1= red].
- All CSPs search algorithms generate successors by considering possible assignments for only a single variable at each node in the search space.
- Adding assignments cannot correct a violated constraint.

Backtracking Search For CSPs

Basic idea: backtracking search uses depth first search choosing values for one variable at a time and backtracks when a variable has no legal values left to assign.

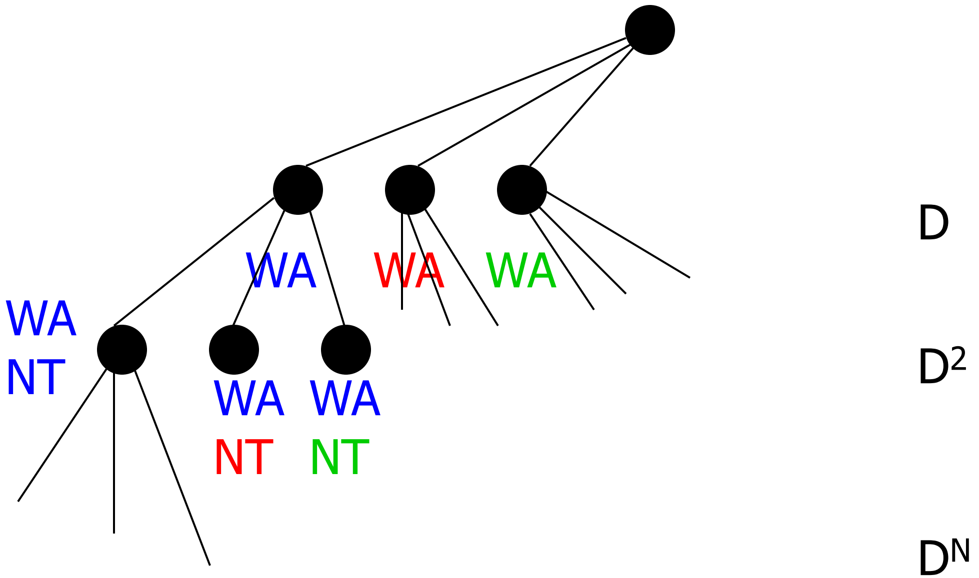
- Depth-first search for CSPs with single-variable assignments and backtracking is called backtracking search.

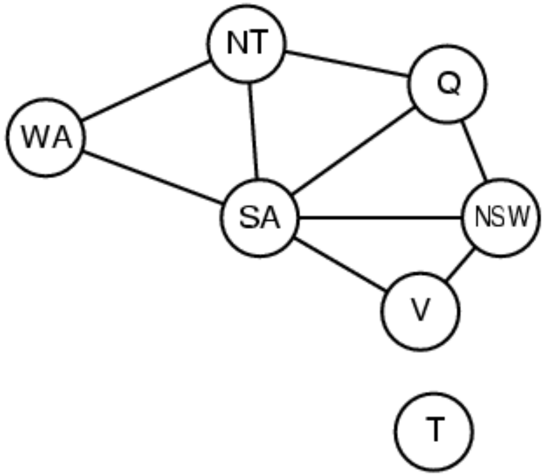
- Backtracking search is the basic uninformed algorithm for CSPs.
- **Policy:** when a branch of the search fails, search backs up to the preceding variable and tries a different value for it. This is called *chronological backtracking* because the most recent decision point is revisited.

Backtracking Search

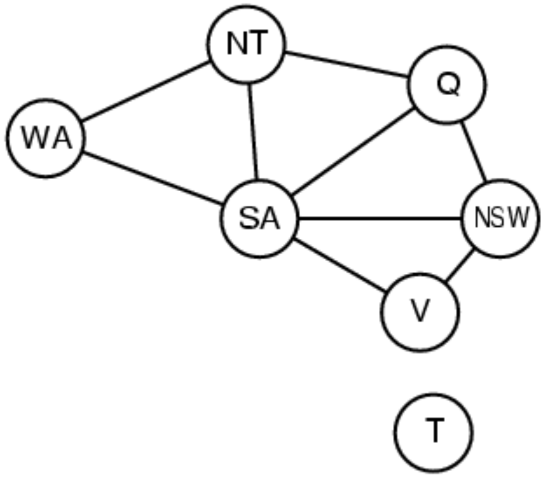
- Special property of CSPs: They are commutative; This means: the order in which we assign variables does not matter. NT WA = WA NT

- Better search tree: First order variables, then assign them values one-by-one.

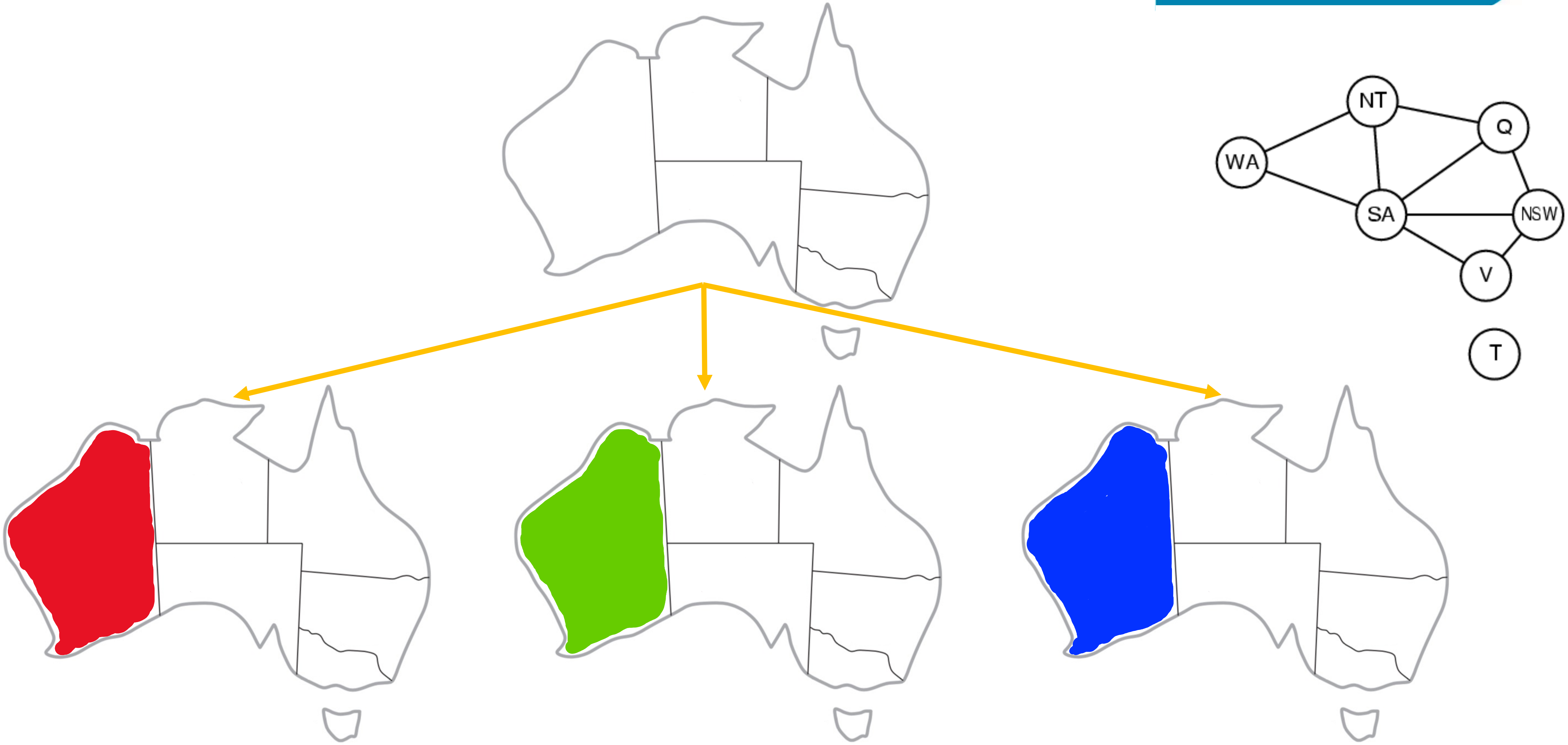




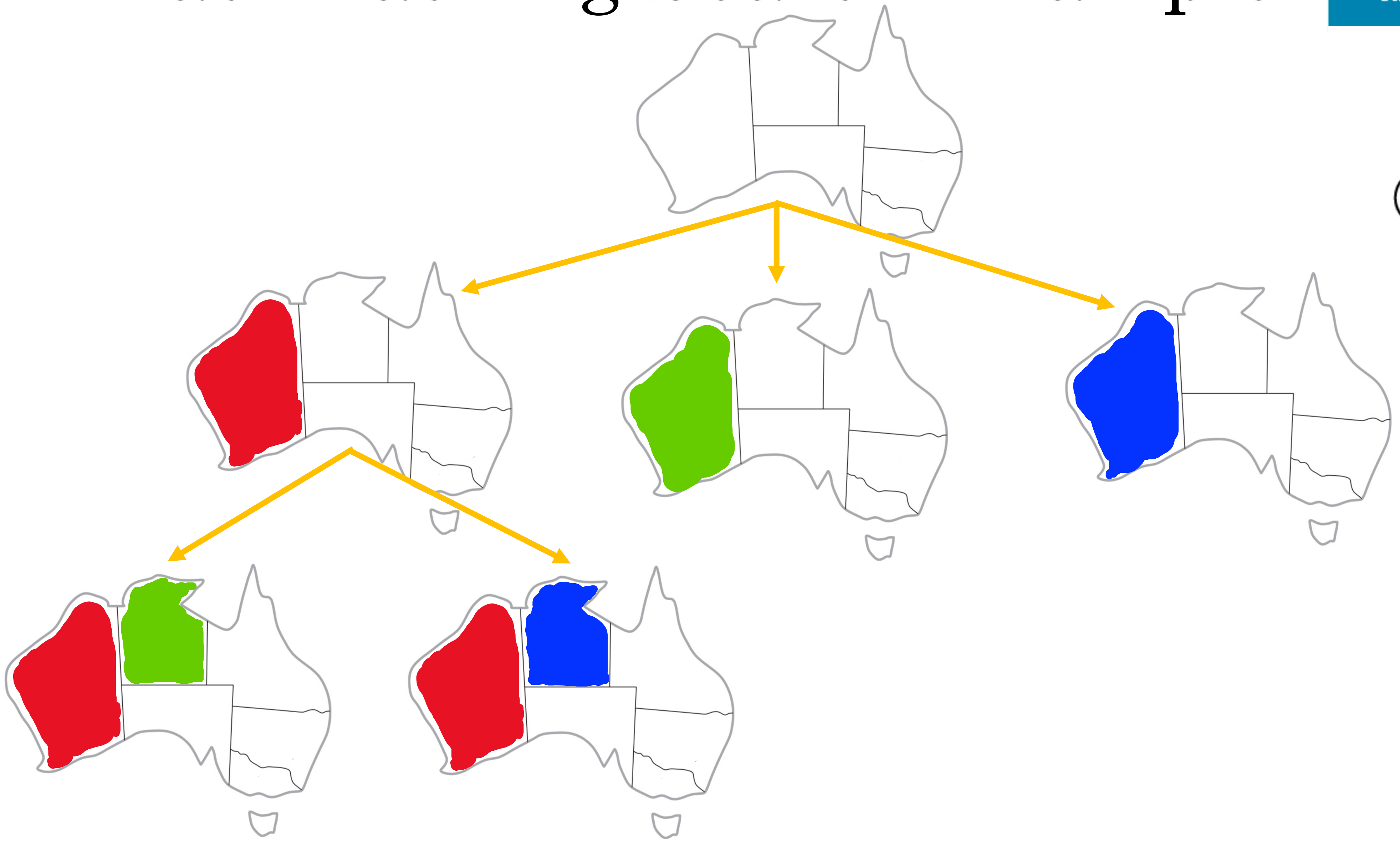
Backtracking Search Example

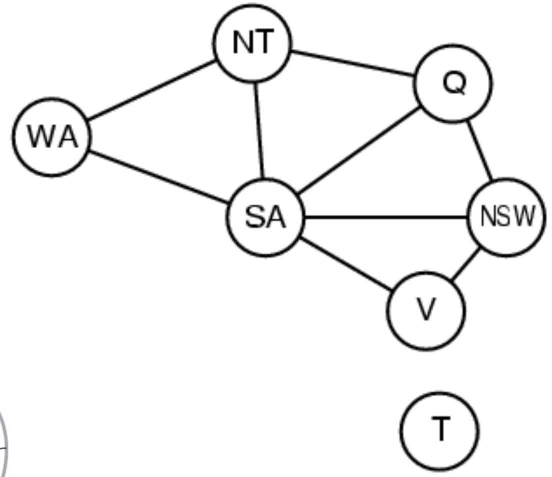


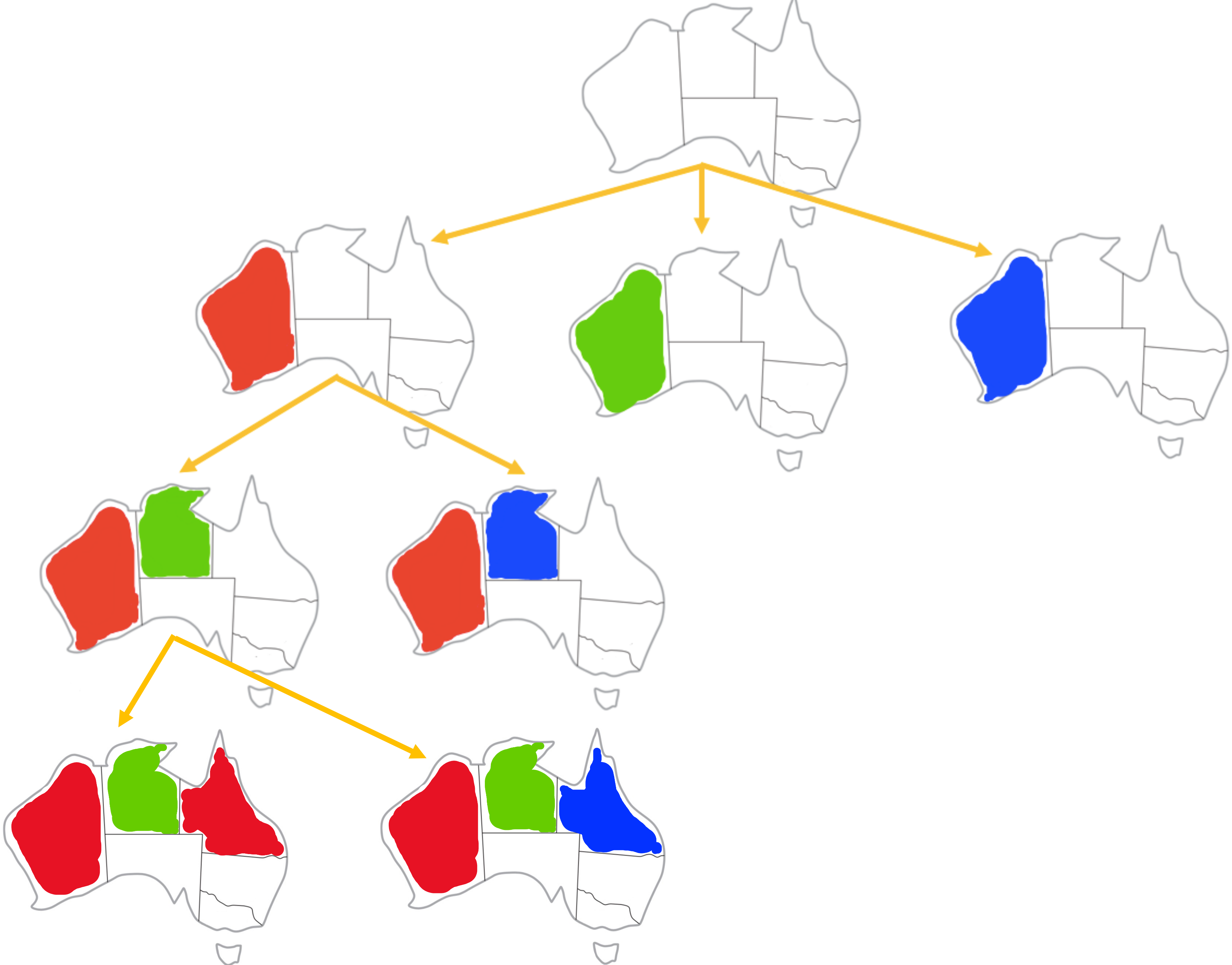
Backtracking Search Example



Backtracking Search Example







Constraint Satisfaction Problems

Notice: Standard representation → no need for domain specific initial state, successor function or goal test.

- SELECT-UNASSIGNED-VARIABLE and ORDER-DOMAIN-VALUES can be used to implement the general purpose heuristics.

- This algorithm is not effective for large problems.

- Improvements: can be achieved if the following questions are addressed:

Constraint Satisfaction Problems

- Which variable should be assigned next and in what order should its values be tried?
- What are the implication of the current variable assignments for the other UNASSIGNED variables?
- When a path fails, can the search avoid repeating this failure in subsequent paths?

CSP Variable & Value Ordering

var ← SELECT-UNASSIGNED-VARIABLE (variable [csp], assignment, csp)

- This statement simply selects the next unassigned variable in the order given by the list variable [csp].

- It seldom results in efficient search.

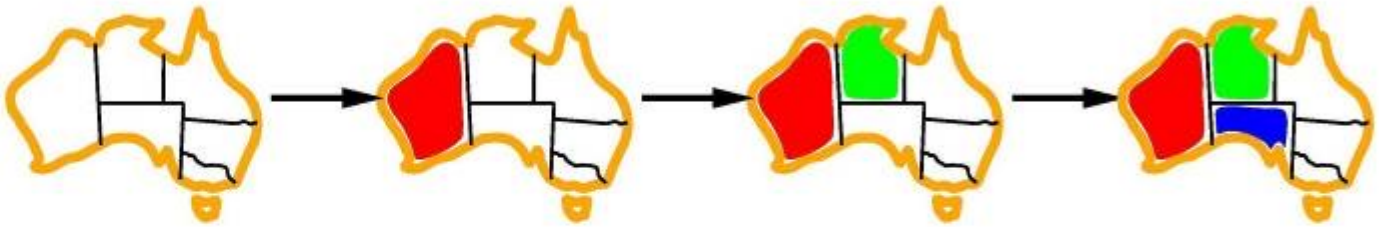
- **<u>Solution</u>:** Choose variable with the fewest "legal" values

→ Minimum Remaining Value (MRV heuristic) also called most constrained variable.

Notice: if there is a variable X with zero legal values remaining, the MRV heuristic will select X and failure will be detected immediately avoiding *pointless search* through other variables which always will fail when X is finally selected.

Constraint Satisfaction Problems

- Most constrained variable: choose the variable with the fewest legal values



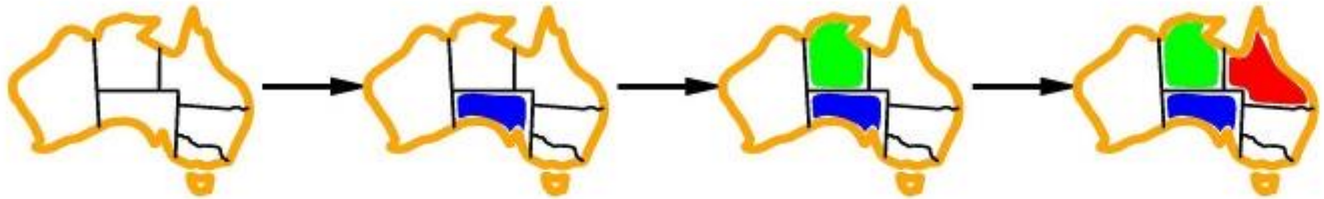
- Example WA = red, NT = green → SA = blue rather than assigning Q.

- After assigning SA, values for Q, NSW and V are all forced.

- **The performance is better than simple backtracking.**

Constraint Satisfaction Problems

Tie breaker among most constrained variables : most constraining variable; choose the variable with the most constraint on remaining variables



Degree heuristic: What is the first region to color?

Idea: Choose the variable that is involved in the largest number of constraints on other unassigned variables.

<u>Example</u>: degree heuristic for SA is 5.

Constraint Satisfaction Problems

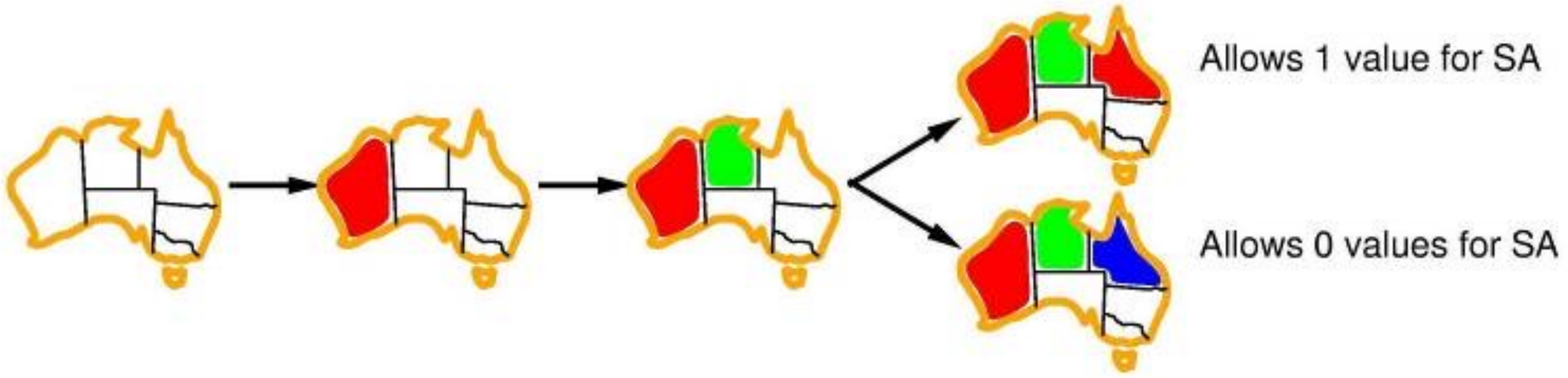
- **Least constraining value (LCV):** Once a variable is selected, how to decide on the **order** in which to examine the values?

- **Solution:** Choose the least constraining value so that to leave maximum flexibility for subsequent variable assignments.

- **Example:** WA=red, NT=green, choosing blue for Q is a bad choice because it eliminates the last legal value for SA.

Constraint Satisfaction Problems

- Least constraining value: the one that rules out the fewest values in the remaining variables



Inference In CSPs

- Inference in CSPs: propagating Information through constraints

- **Key Idea:** Instead of considering the constraints on a variable **only at the time** that the variable is chosen by SELECT-UNASSIGNED-VARIABLE, LOOK at some constraint **earlier or even before.**

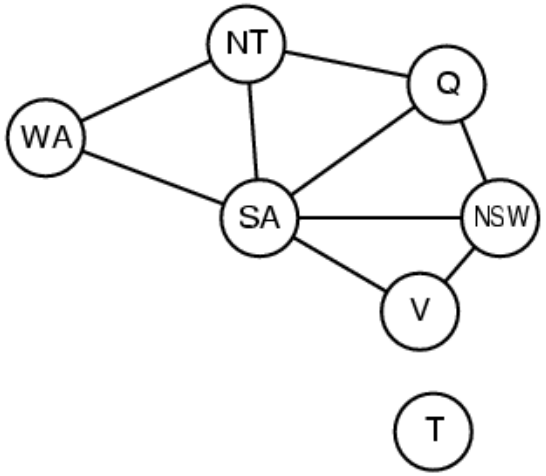
- One alternative: **Forward Checking** (FC).

- **Forward Checking** looks at each unassigned variable ***Y*** that is connected to ***X*** by a constraint and deletes from ***Y***'s domain any value that is **inconsistent** with the value chosen for ***X***.

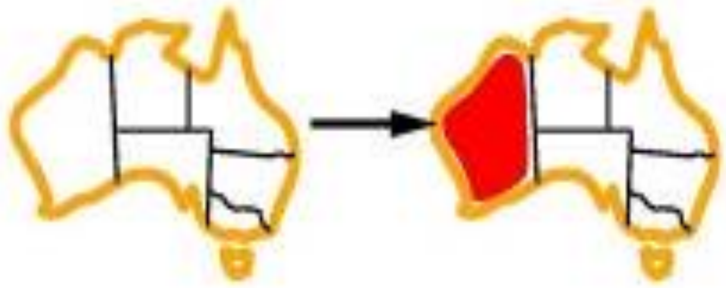
- Forward Checking is one of the **simplest forms of inference.**

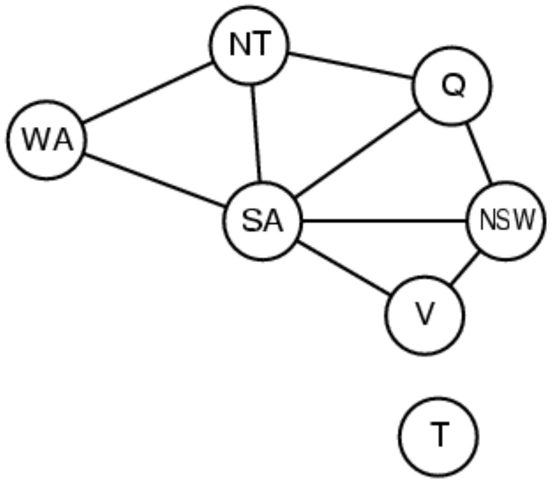
Forward Checking

- **Key idea:** keep track of remaining legal values for unassigned variables, terminate search when any variable has no legal values

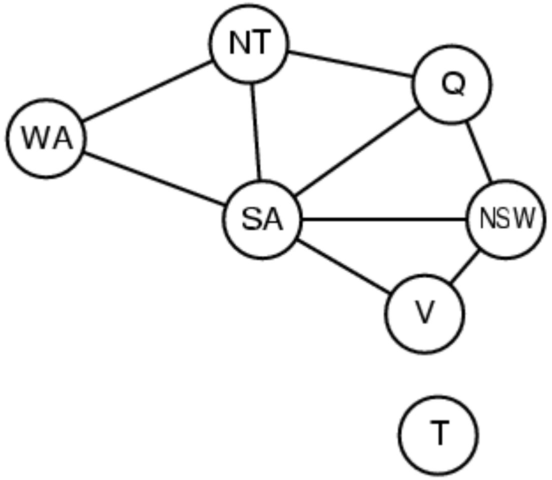


Forward Checking

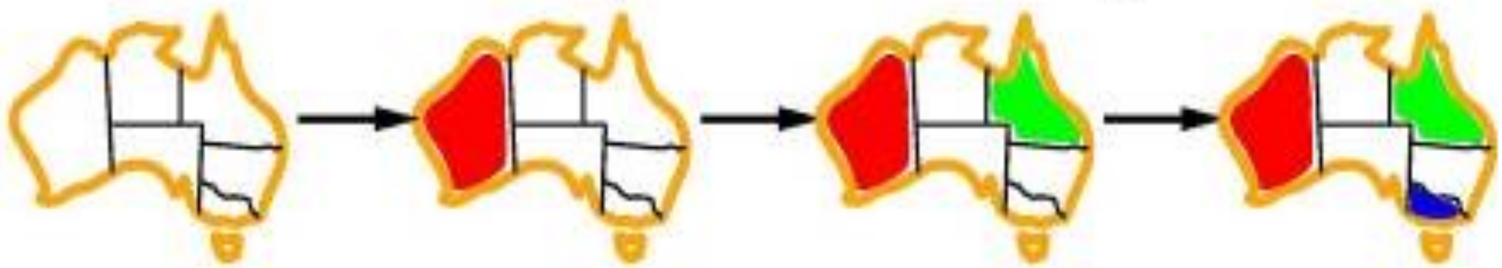


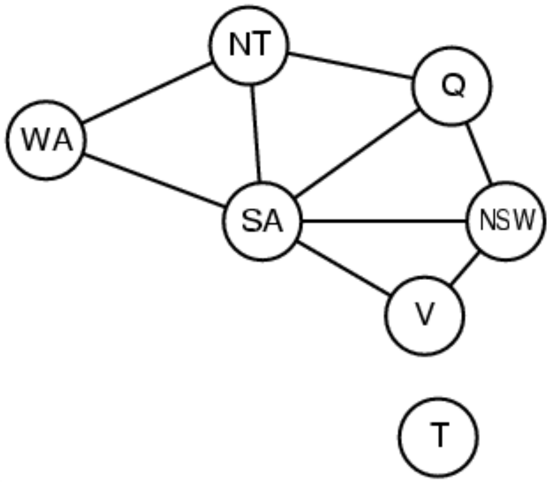


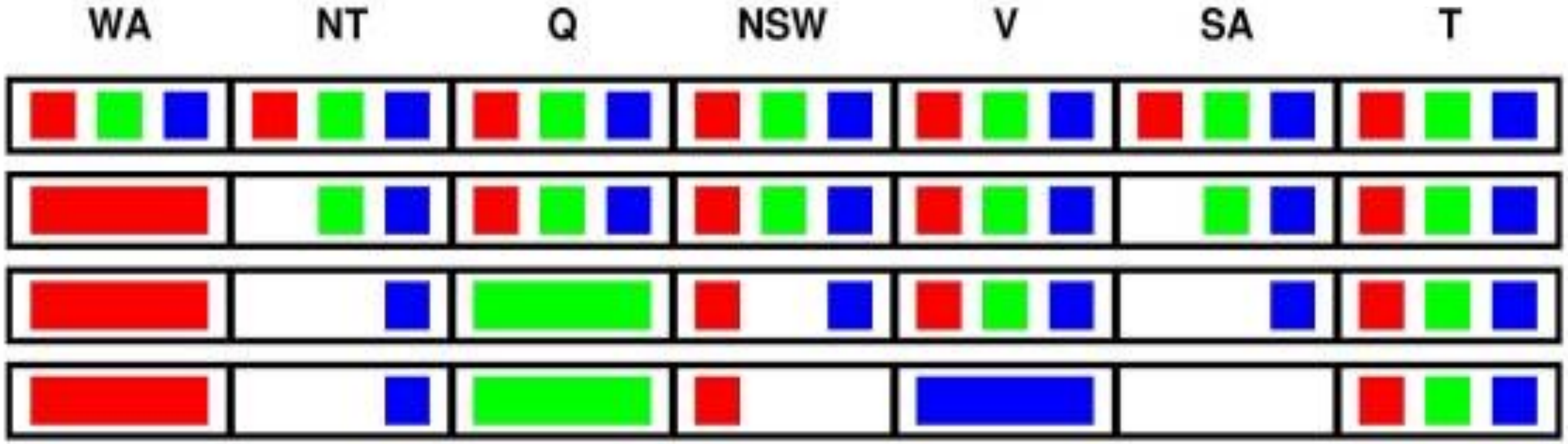
Forward Checking



Forward Checking







Constraint Satisfaction Problems

- After WA= red and Q= green, NT and SA with simple value. → selection by MRV
  - FC computes the information that the MRV heuristic needs to do its job.

- After V = blue, FC detects that the partial assignment {WA=red, Q=green, V=blue} is inconsistent → the algorithm will therefore backtracks immediately.

	WA	*NT*	*Q*	*NSW*	*V*	*SA*	*T*
Initial domain	RGB	RGB	RGB	RGB	RGB	RGB	RGB
After *WA=red*	R	GB	RGB	RGB	RGB	GB	RGB
After *Q=green*	R	B	G	R B	RGB	B	RGB
After *V=blue*	R	B	G	R	B		RGB

CSP: Constraint Propagation

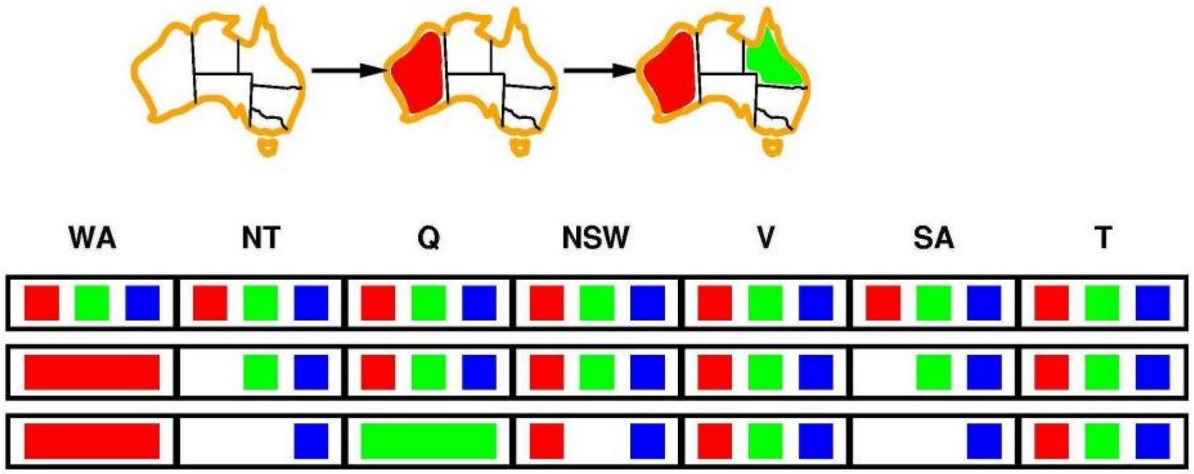
- **Problem with FC**: cannot detect all inconsistencies.
- **Example:** WA=red, Q=green → NT and SA are forced to be blue but they are adjacent. FC does not detect this as an inconsistency.

- **Solution:** Implications on one variable onto other variables should be propagated. → Arc consistency.

- **Requirements:**
  - do this fast.
  - Time for propagating constraints should not be greater than reducing the amount of search.

Constraint Propagation

- Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures



- E.g.: NT and SA cannot both be blue
- Constraint propagation repeatedly enforces constraints locally.

FC

Arc Consistency (AC)

- What is an arc? A directed link between variables in the constraint graph.

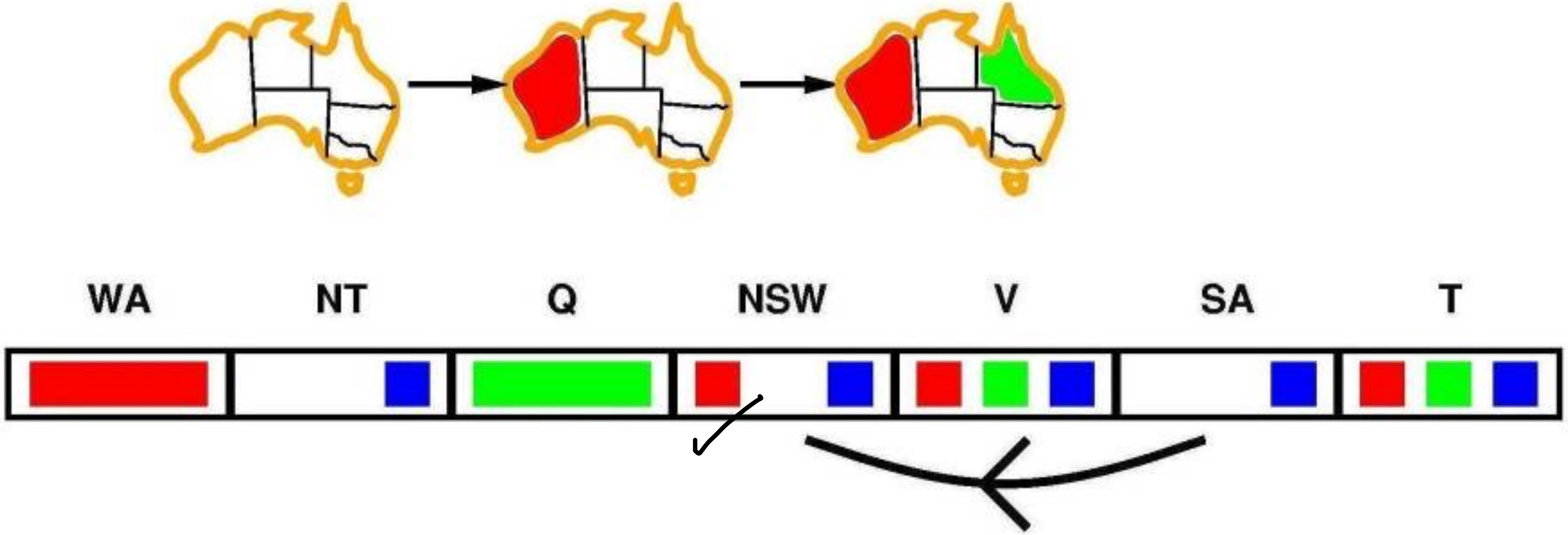
- **Definition:** Given the current domains of SA and NSW, the arc is consistent if, for every value x of SA there is some value y of NSW that is consistent with x.
- Example: SA={B}, NSW={R, B} $\forall x$ $\exists y$
  - The arc SA→ NSW is consistent
  - The arc NSW→ SA is not consistent.

- This technique try to modify an existing constraint satisfaction problem such that the search space can be reduced significantly.
- AC can be applied as a preprocessing before the beginning of the search process or during the search as a propagation step after every assignment.

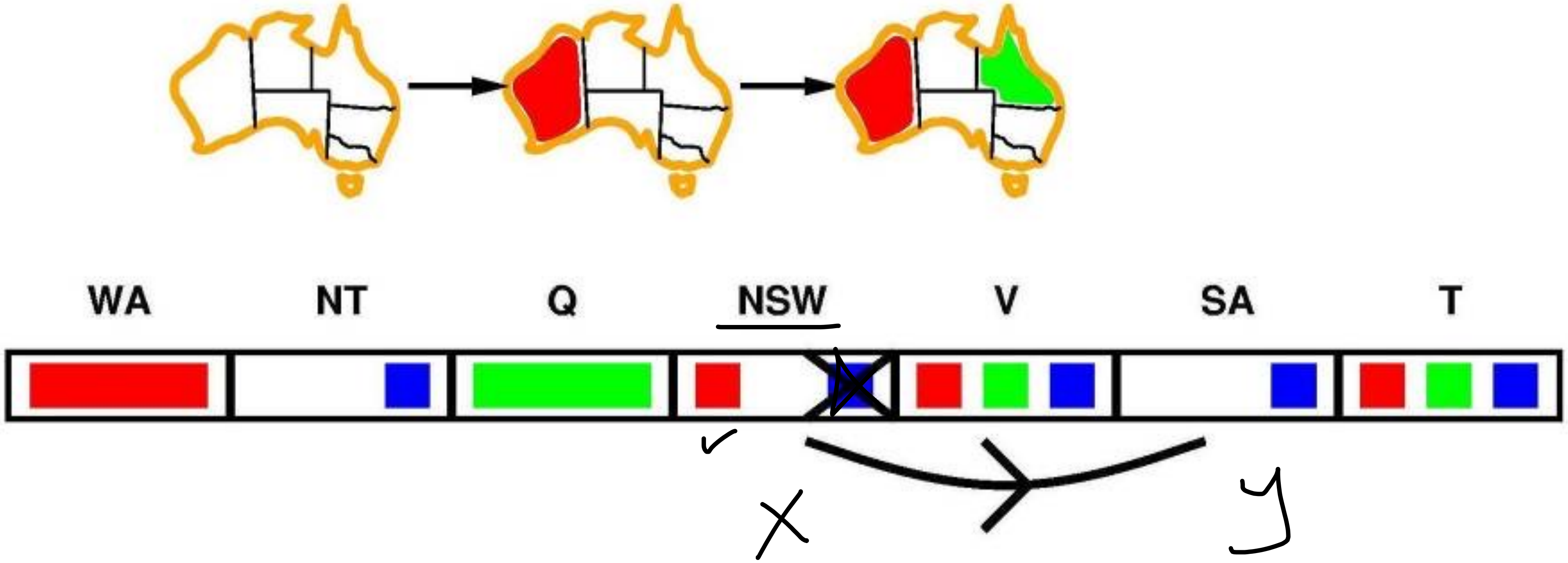
Arc Consistency

- Simplest form of propagation makes each arc consistent
- X → Y is consistent *iff* for every value *x* of X there is some allowed *y*

$\forall x$ $\exists y$

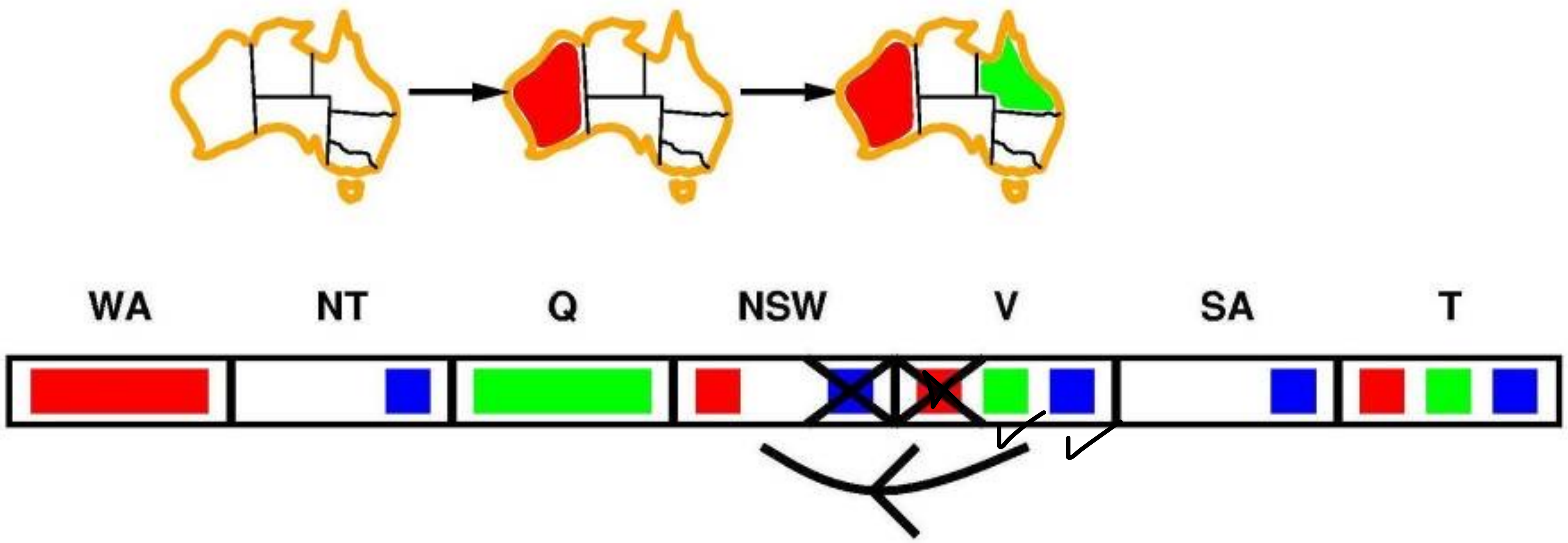


Arc consistency



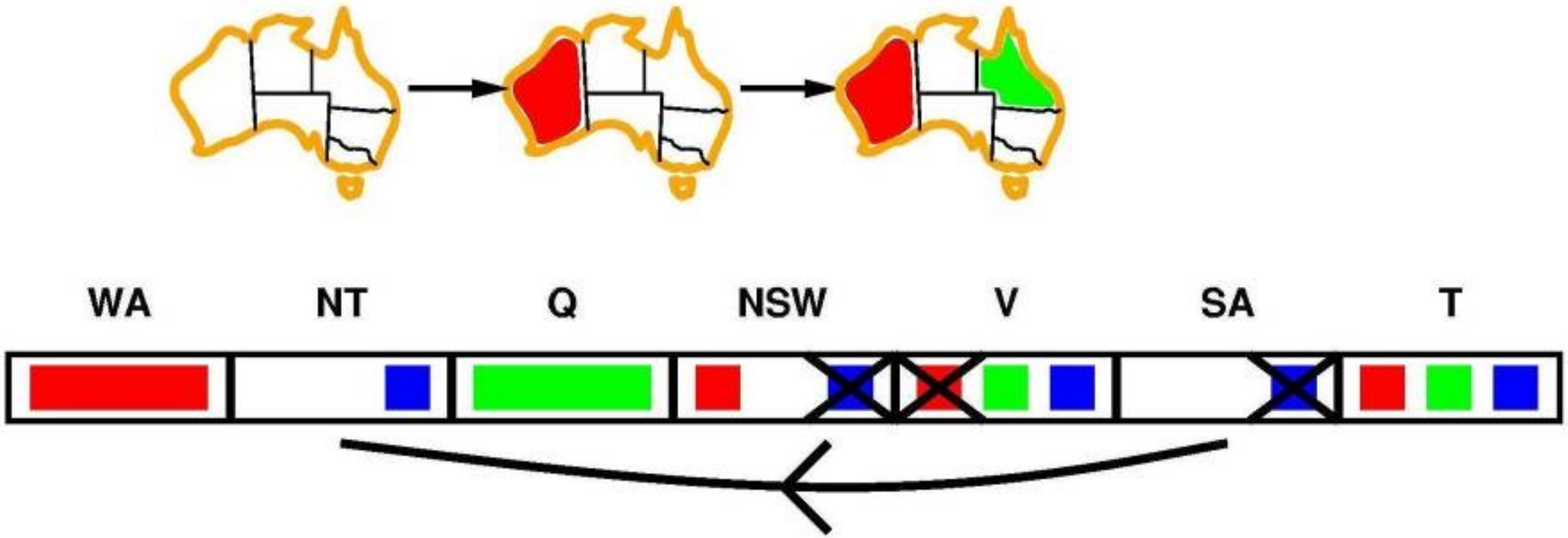
Arc consistency

- If X loses a value, neighbors of X need to be rechecked



Arc consistency

- Arc consistency detects failure earlier than forward checking, Can be run as a preprocessor or after each assignment.



Arc Consistency Algorithm AC-3

جامعة الملك سعود
King Saud University

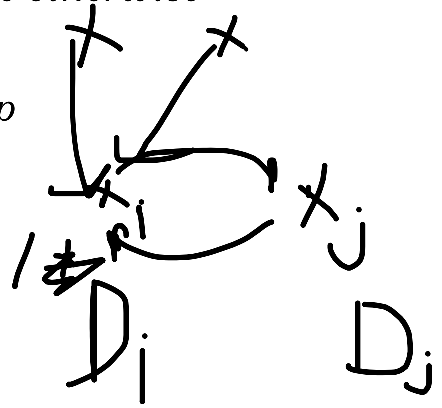
function AC-3(csp) ***returns*** *false if an inconsistency is found and true otherwise*

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**
(Xi, Xj) ← *REMOVE-FIRST(queue)*
if REVISEE(*csp, Xi, Xj)* ***then***
if size of Di = 0 then return *false*
for each *Xk in Xi.NEIGHBORS –{ Xj} do add (Xk, Xi) to queue*

return *true*



---

function REVISE(*csp, Xi, Xj) returns true iff we revise the domain of Xi,*

revised ← *false*

for each *x* ***in*** *Di* ***do***

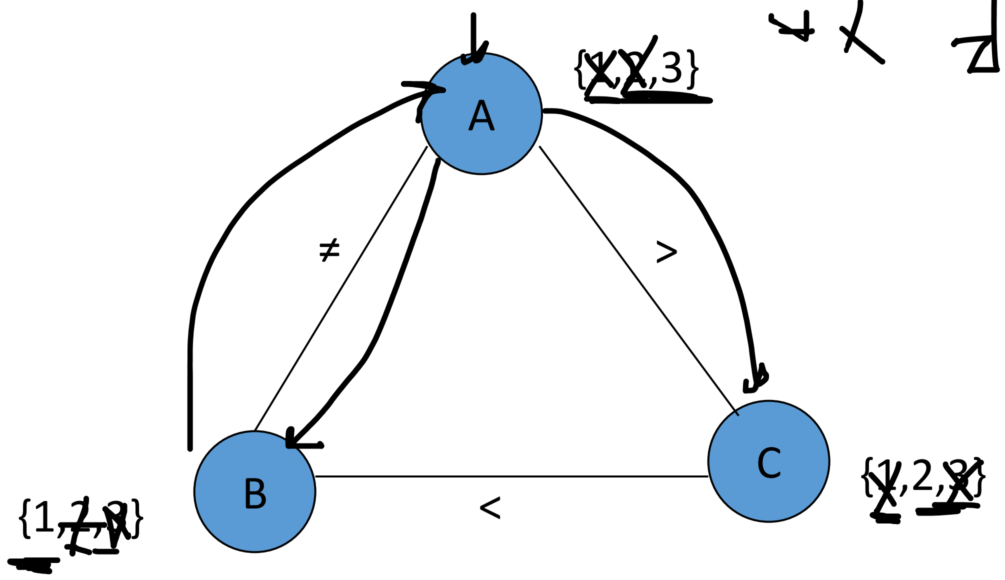
if no value y in *Dj allows (x , y) to satisfy the constraint between Xi and Xj* ***then***

delete *x from Di*

revised ← true

Return *revised*

CSP Example



Arcs	A	B	C	Added arcs
~~AC~~	{2,3}	=	=	BA
~~CA~~	=	=	{1,2}	BC
AB	=	=	=	
BA	=	=	=	
BC	=	{1}	=	AB
CB	=	=	{2}	**AC**
BA	=	=	=	
BC	=	=	=	
AB	=	=	=	
AC	{3}	=	=	BA
BA	=	=	=	

Em

Backtracking algorithm for CSPs

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)* MRV degree

  for each *value* **in** *ORDER-DOMAIN-VALUES(var, assignment, csp)* **do** LCV

    if value is consistent with assignment then

      add { var = *value} to assignment*

      inferences ← INFERENCE(csp, var, value) FC or AC

      if inferences ≠ failure then

        add inferences to assignment

      Result ← *BACKTRACK(assignment, csp)*

      if *result ≠ failure* ***then***

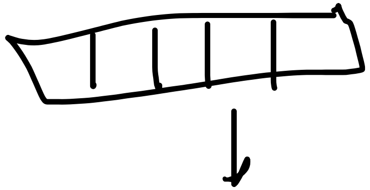
        return *result*

    remove { *var = value* } ***and inferences*** *from assignment*

  return *failure*

Local Search For CSPs

- Many CSPs can be solved efficiently using local search algorithms.
- They use complete-state formulation.
- Initial state assigns a value to every variable and the successor function works by changing the value of one variable at a time.
- In choosing a new value for a variable, the most obvious heuristic is to select the value that results in the minimum number of conflicts with other variables: « The Min-Conflicts » heuristic.



MIN-CONFLICTS Algorithm For Cpss By Local Search

Function Min-Conflicts(csp, max_steps) **returns** a solution or failure
inputs: csp, a constraint satisfaction problem
max_steps, the number of steps allowed before giving up.

current ← an initial complete assignment for csp
For i=1 to max_steps **do**
If current is a solution for csp **then return** current
var ← a randomly chosen conflicted variable from csp.VARIABLES
value← the value v for var that **minimizes** CONFLICTS(var,v, current, csp)
Set var = value in current
return failure

Notice : Local search is **very effective** for reasonable initial state.

Describe an example of a real world CSP, the solution it presented, as well as its local and global impact

Hand it in, on LMS.