Künstliche Intelligenz
Constraint Satisfaction Problems

Dr.-Ing. Stefan Lüdtke

Universität Leipzig

Center for Scalable Data Analytics and Artificial Intelligence (ScaDS.AI)

Problemlösung durch Suchen

Problemlösende Agenten

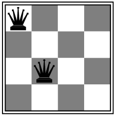
- Ein *problemlösender Agent* ist ein Spezialfall des zielbasierten Agenten
- Sucht nach Aktionsfolgen, die einen wünschenswerten Zustand erreichen.
- Varianten der Suche:
  - *nichtinfomierte* Suche: Keine Information über den „Abstand“ zwischen wünschenswertem Zustand und anderen Situationen.
  - *informierte* Suche: Abstandsinformation vorhanden.
- Was bedeutet hierbei „Problem“ und „Lösung“?

Constraint satisfaction problems (CSPs)

- Standard-Suchproblem: Zustand ist “Black Box” (beliebige Datenstruktur)
- CSP: Zustand ist definiert über *Variablen* V_i mit *Werten* aus *Domäne* D_i
- Zieltest ist Menge von *Constraints*(Bedingungen), die die erlaubten Kombinationen von Variablen für eine Teilmenge der Variablen einschränken
- Erlaubt, effizientere Algorithmen als generelle Suchalgorithmen zu verwenden

4-Damen als CSP

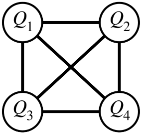
- Eine Dame pro Spalte. In welche Zeile soll jede Dame?
- Variablen: Q_1, Q_2, Q_3, Q_4
- Domänen: $D_i = \{1, 2, 3, 4\}$
- Constraints:
  - $Q_i \neq Q_j$ (nicht in gleicher Zeile)
  - $|Q_i - Q_j| \neq |i - j|$ (nicht auf gleicher Diagonale)



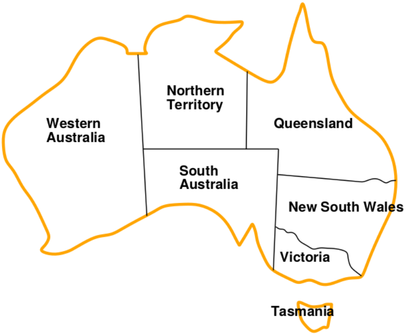
- Constraints können in Menge erlaubter Werte für betreffende Variablen umgewandelt werden, z.B. erlaubte Werte für (Q_1, Q_2) sind $(1, 3)$ $(1, 4)$ $(2, 4)$ $(3, 1)$ $(4, 1)$ $(4, 2)$

Constraint Graph

- Binäres CSP: Jedes Constraint beinhaltet höchstens zwei Variablen
- Constraint Graph: Knoten sind Variablen, Kanten sind Constraints

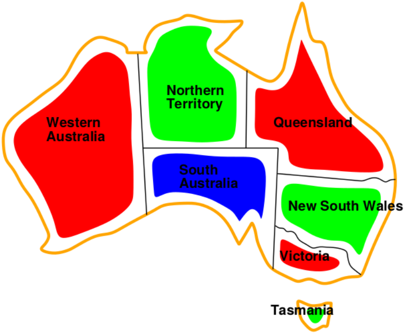


Beispiel: Kartenfärbung



- Variablen: Länder C_i
- Domänen: $\{Rot, Gruen, Blau\}$
- Constraints: $C_1 \neq C_2$, $C_1 \neq C_5$, etc.

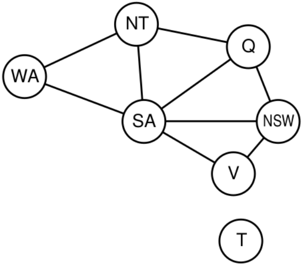
Beispiel: Kartenfärbung



Lösung: Zuweisung von Werten zu jeder Variablen, sodass alle Constraints erfüllt sind, z.B. $WA = red, WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green$

Beispiel: Kartenfärbung

Constraint Graph: Knoten sind Variablen, Kanten sind Constraints



Standard search formulation (incremental)

Let's start with the straightforward, dumb approach, then fix it

States are defined by the values assigned so far

◊ Initial state: the empty assignment, $\{\,\}$

◊ Successor function: assign a value to an unassigned variable
that does not conflict with current assignment.
$\Rightarrow$ fail if no legal assignments (not fixable!)

◊ Goal test: the current assignment is complete

1) This is the same for all CSPs!
2) Every solution appears at depth n with n variables
$\Rightarrow$ use depth-first search
3) Path is irrelevant, so can also use complete-state formulation
4) $b = (n - \ell)d$ at depth ℓ, hence $n!d^n$ leaves!!!!

Backtracking search

Variable assignments are commutative, i.e.,

$[WA = red$ then $NT = green]$ same as $[NT = green$ then $WA = red]$

Only need to consider assignments to a single variable at each node

$\Rightarrow$ $b = d$ and there are d^n leaves

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

Backtracking search is the basic uninformed algorithm for CSPs

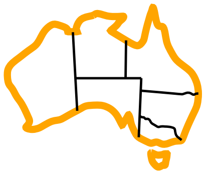
Can solve n-queens for $n \approx 25$

Backtracking search

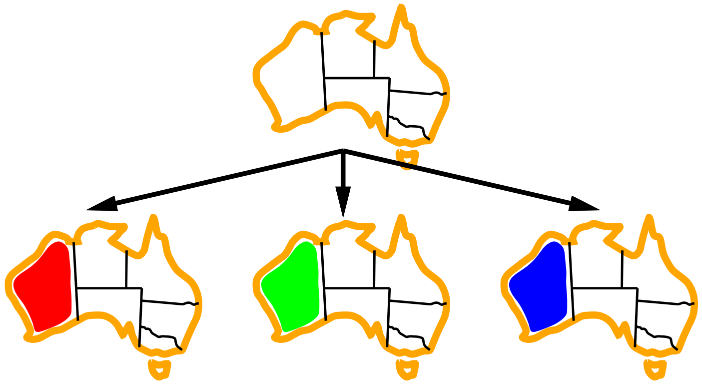
```
function BACKTRACKING-SEARCH(csp) returns solution/failure
   return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
   if assignment is complete then return assignment
   var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
   for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
       if value is consistent with assignment given CONSTRAINTS[csp] then
           add {var = value} to assignment
           result ← RECURSIVE-BACKTRACKING(assignment, csp)
           if result ≠ failure then return result
           remove {var = value} from assignment
   return failure
```

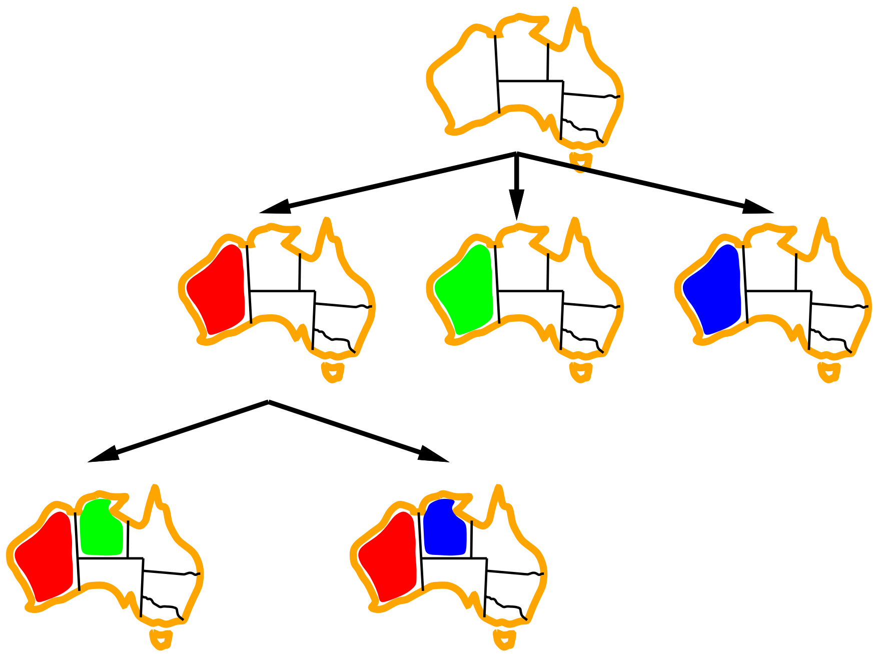
Backtracking example



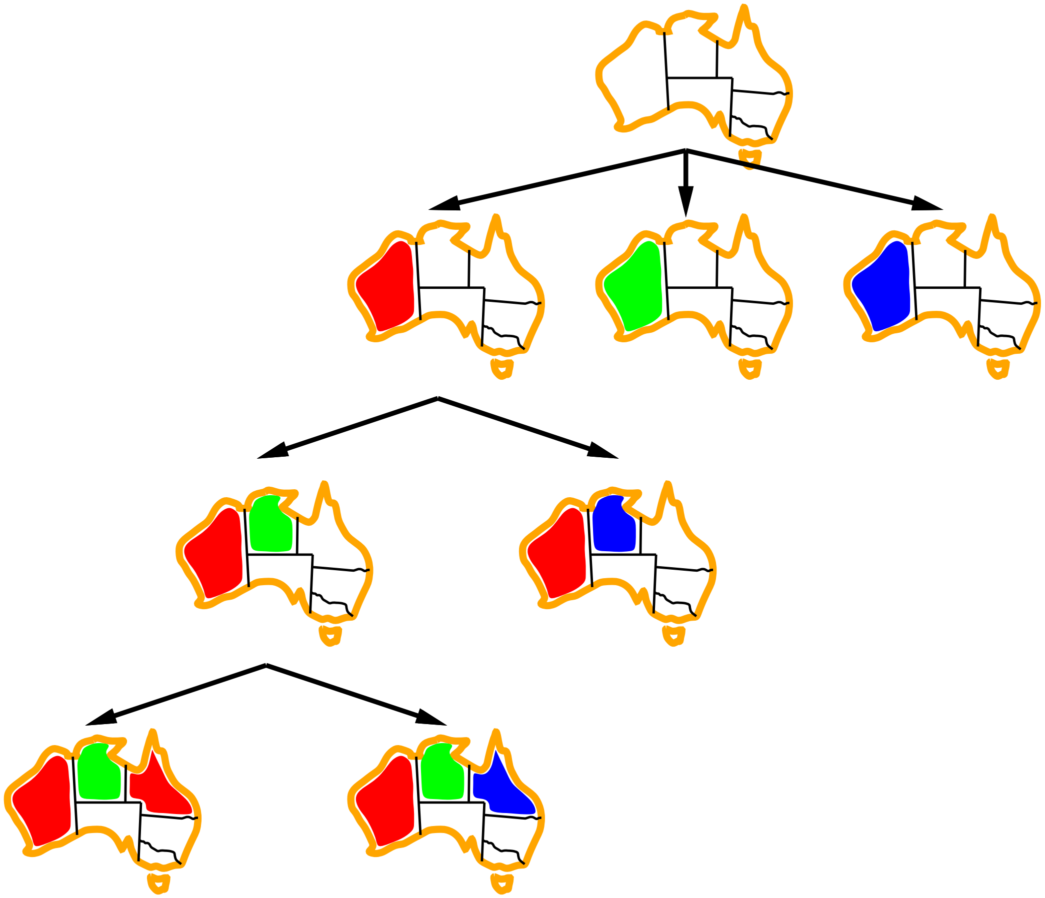
Backtracking example



Backtracking example



Backtracking example



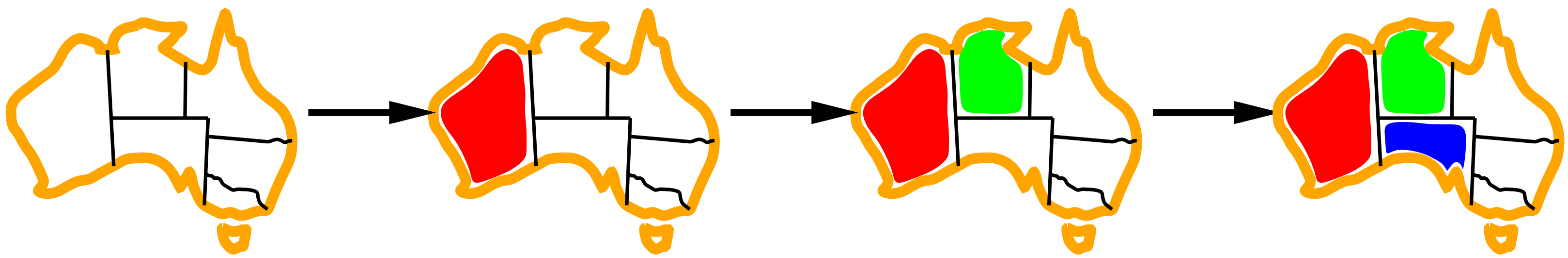
Improving backtracking efficiency

General-purpose methods can give huge gains in speed:

1. Which variable should be assigned next?
2. In what order should its values be tried?
3. Can we detect inevitable failure early?
4. Can we take advantage of problem structure?

Minimum remaining values

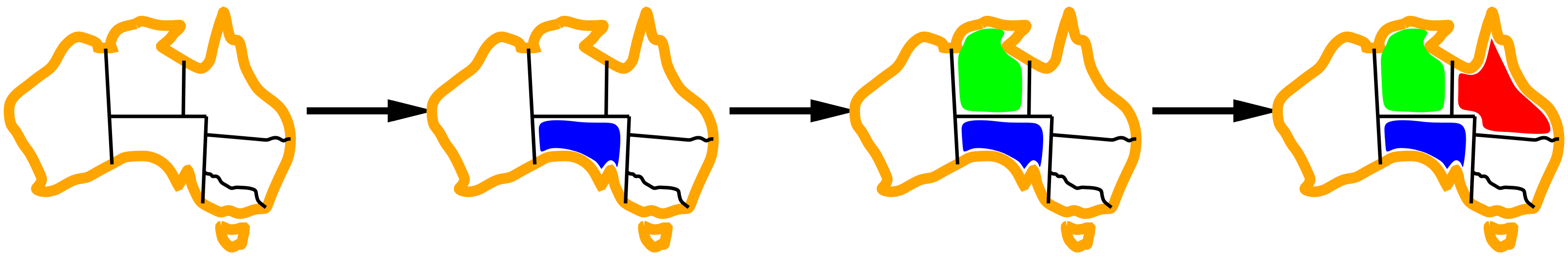
Minimum remaining values (MRV):
choose the variable with the fewest legal values



Degree heuristic

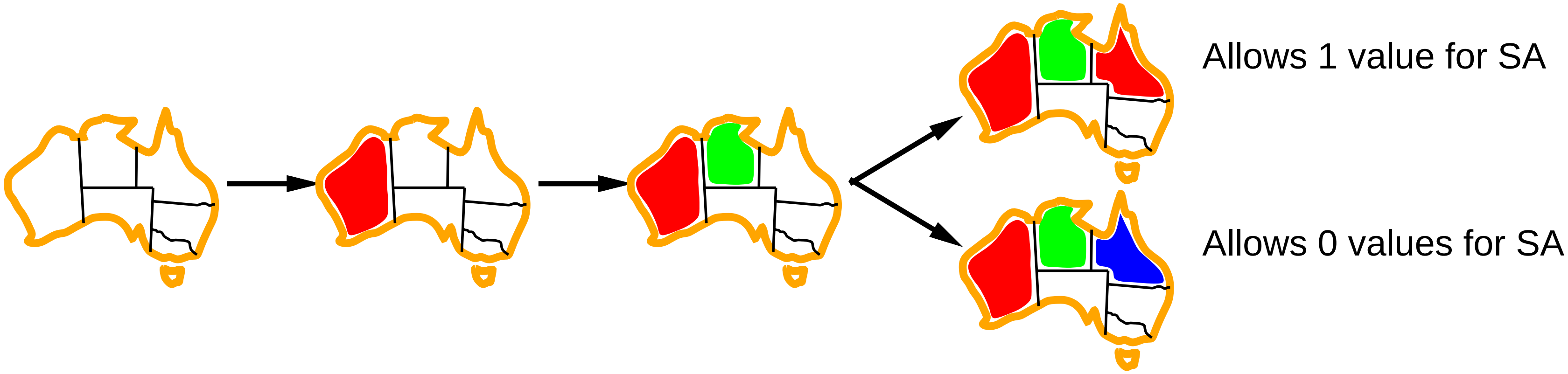
Tie-breaker among MRV variables

Degree heuristic:
choose the variable with the most constraints on remaining variables



Least constraining value

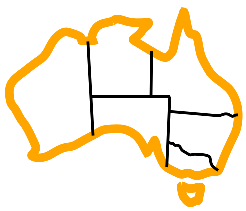
Given a variable, choose the least constraining value:
the one that rules out the fewest values in the remaining variables

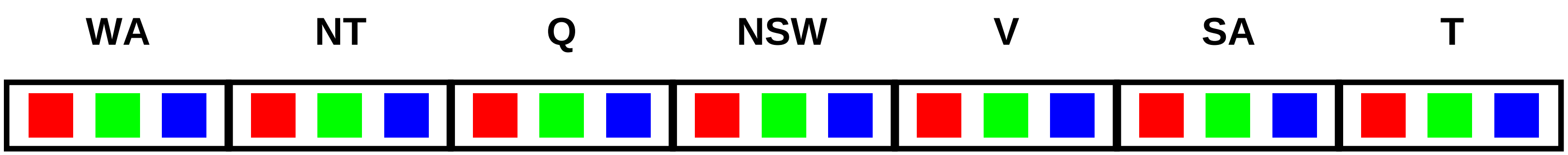


Combining these heuristics makes 1000 queens feasible

Forward checking

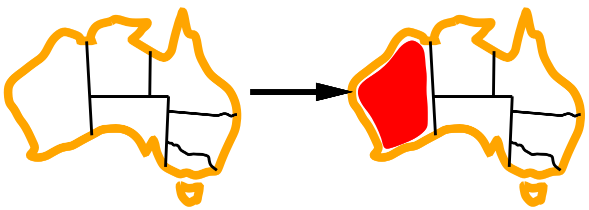
Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values

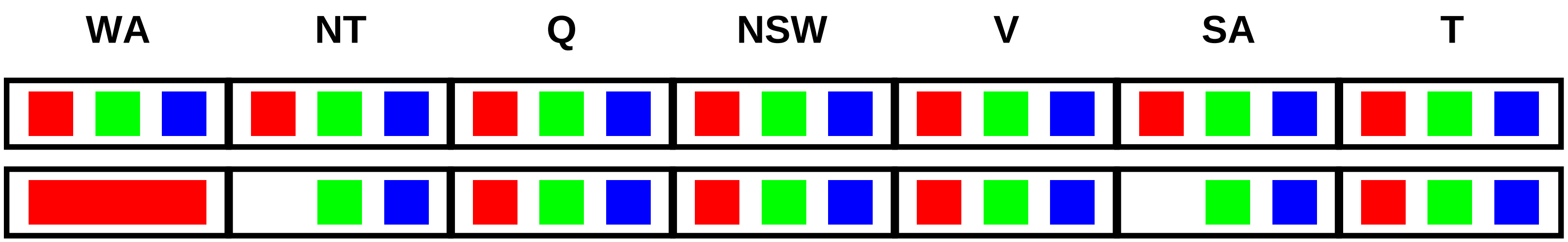




Forward checking

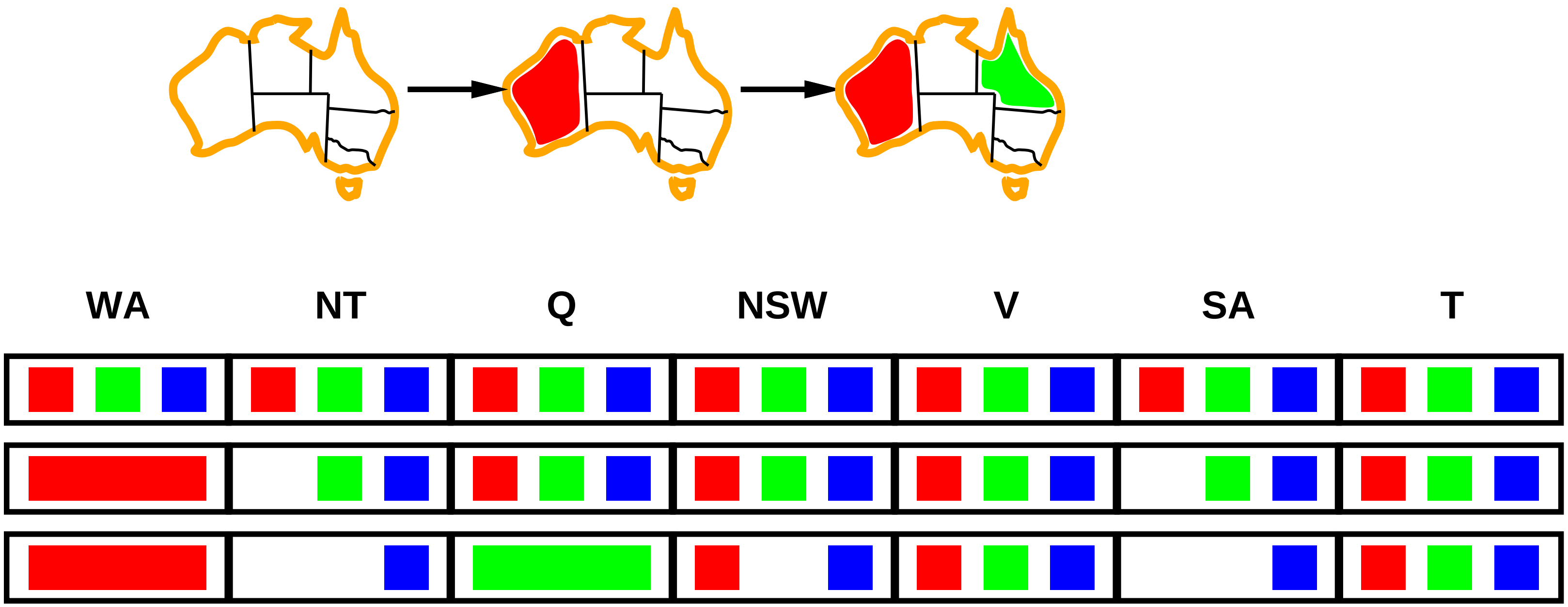
Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values





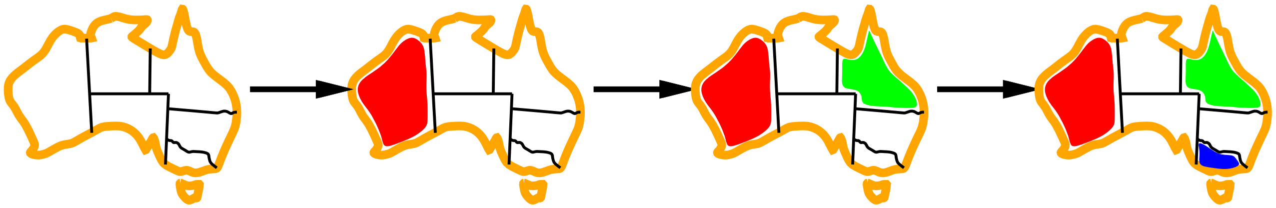
Forward checking

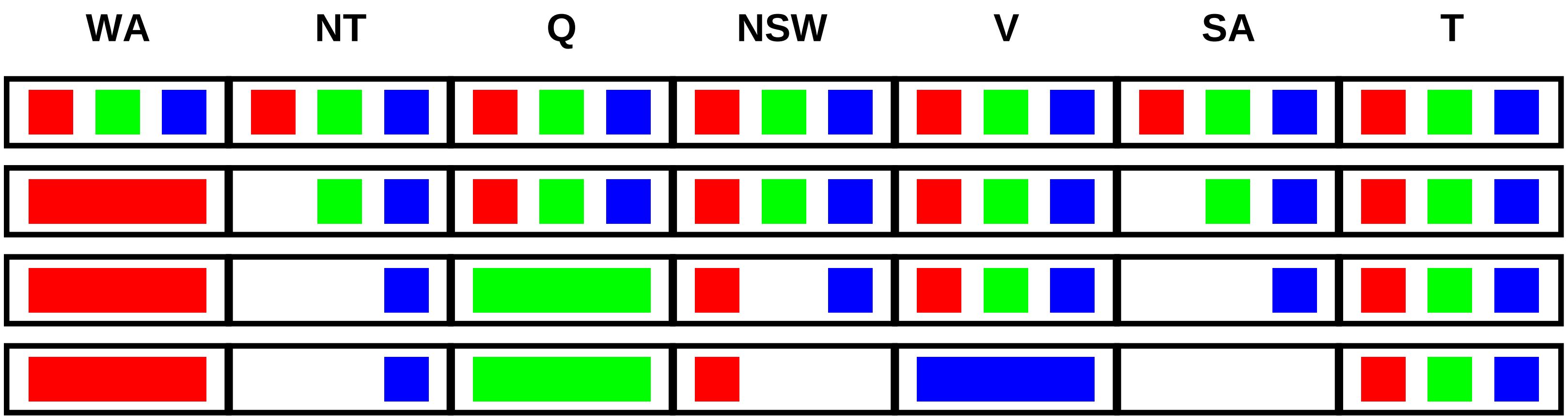
Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values



Forward checking

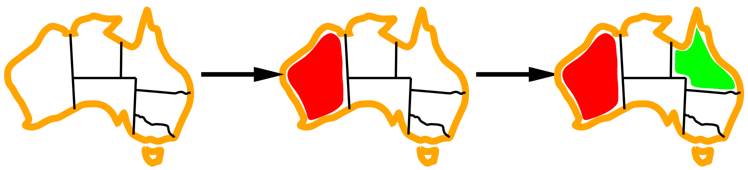
Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values





Constraint propagation

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



NT and SA cannot both be blue!

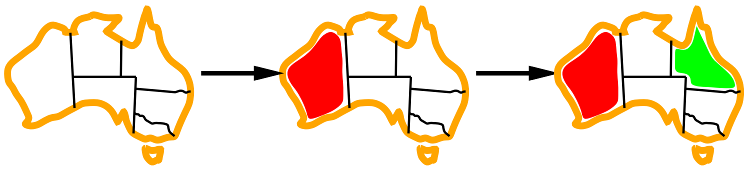
Constraint propagation repeatedly enforces constraints locally

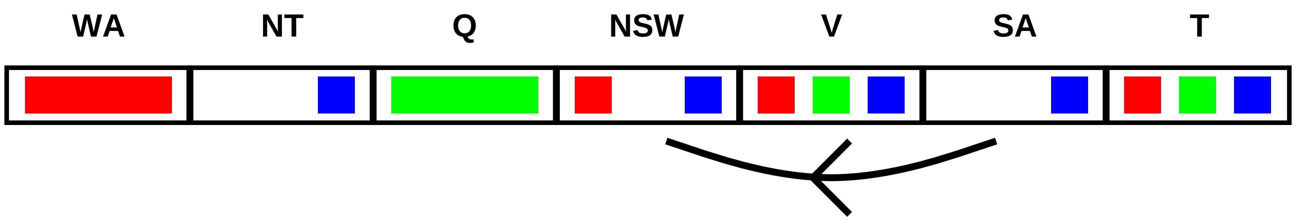
Arc consistency

Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent iff

for **every** value x of X there is **some** allowed y



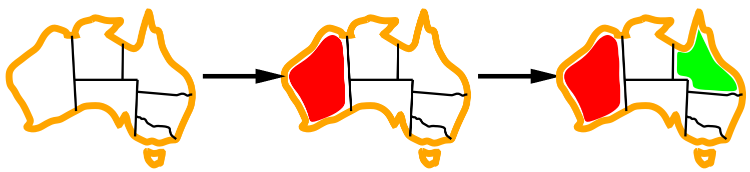


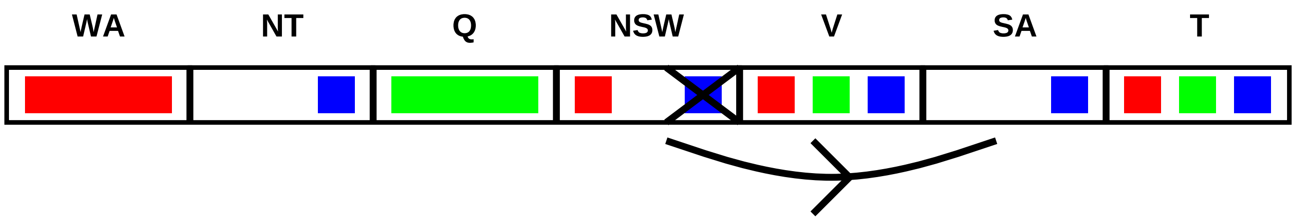
Arc consistency

Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent iff

for **every** value x of X there is **some** allowed y



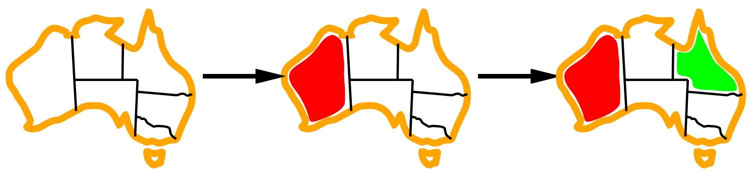


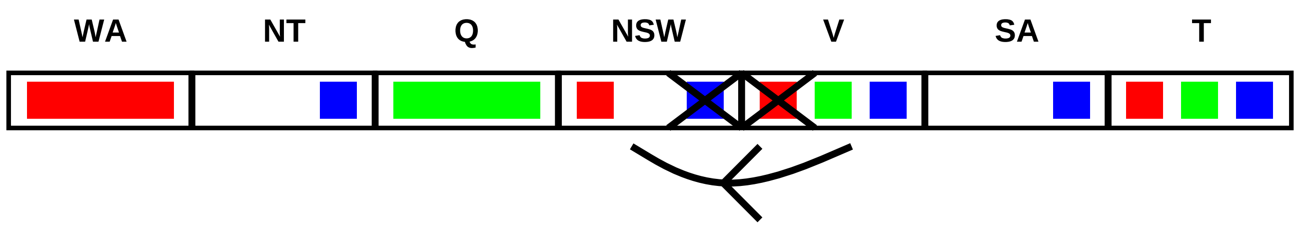
Arc consistency

Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent iff

for **every** value x of X there is **some** allowed y





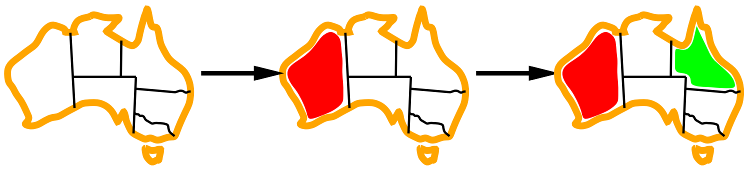
If X loses a value, neighbors of X need to be rechecked

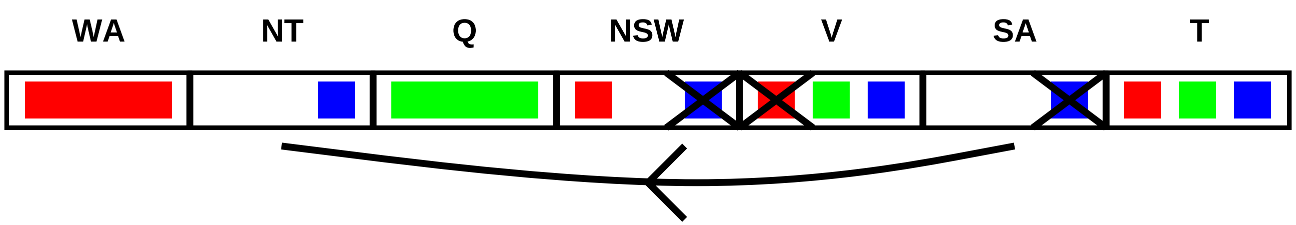
Arc consistency

Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent iff

for **every** value x of X there is **some** allowed y





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

Arc consistency detects failure earlier than forward checking

Can be run as a preprocessor or after each assignment

Arc consistency algorithm

```
function AC-3( csp) returns the CSP, possibly with reduced domains
   inputs: csp, a binary CSP with variables {X_1, X_2, ..., X_n}
   local variables: queue, a queue of arcs, initially all the arcs in csp

   while queue is not empty do
      (X_i, X_j) ← REMOVE-FIRST(queue)
      if REMOVE-INCONSISTENT-VALUES(X_i, X_j) then
         for each X_k in NEIGHBORS[X_i] do
            add (X_k, X_i) to queue
```

```
function REMOVE-INCONSISTENT-VALUES( X_i, X_j) returns true iff succeeds
   removed ← false
   for each x in DOMAIN[X_i] do
      if no value y in DOMAIN[X_j] allows (x,y) to satisfy the constraint X_i ↔ X_j
         then delete x from DOMAIN[X_i];  removed ← true
   return removed
```

$O(n^2d^3)$, can be reduced to $O(n^2d^2)$ (but detecting **all** is NP-hard)

Summary

CSPs are a special kind of problem:
states defined by values of a fixed set of variables
goal test defined by constraints on variable values

Backtracking = depth-first search with one variable assigned per node

Variable ordering and value selection heuristics help significantly

Forward checking prevents assignments that guarantee later failure

Constraint propagation (e.g., arc consistency) does additional work
to constrain values and detect inconsistencies

The CSP representation allows analysis of problem structure

Tree-structured CSPs can be solved in linear time

Iterative min-conflicts is usually effective in practice