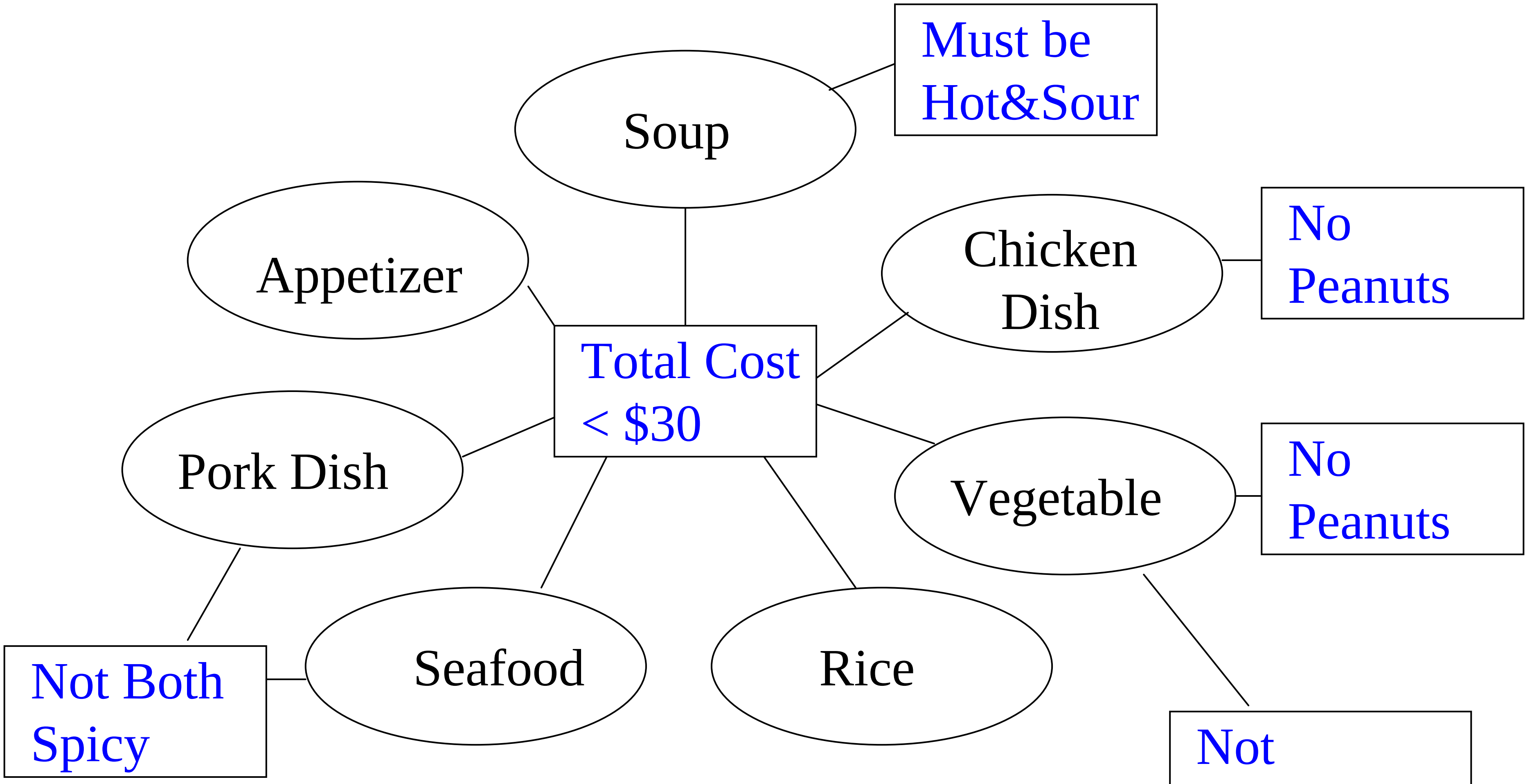
Constraint Satisfaction Problems



Constraint Network

Formal Definition of CSP

- A constraint satisfaction problem (CSP) is a triple (V, D, C) where
  - V is a set of variables $X_1, \dots, X_n$.
  - D is the union of a set of domain sets $D_1,\dots,D_n$, where D_i is the domain of possible values for variable X_i.
  - C is a set of constraints on the values of the variables, which can be pairwise (simplest and most common) or k at a time.

CSPs vs. Standard Search Problems

- Standard search problem:
  - state is a "black box“ – any data structure that supports successor function, heuristic function, and goal test
- CSP:
  - state is defined by variables X_i with values from domain D_i
  - goal test is a set of constraints specifying allowable combinations of values for subsets of variables

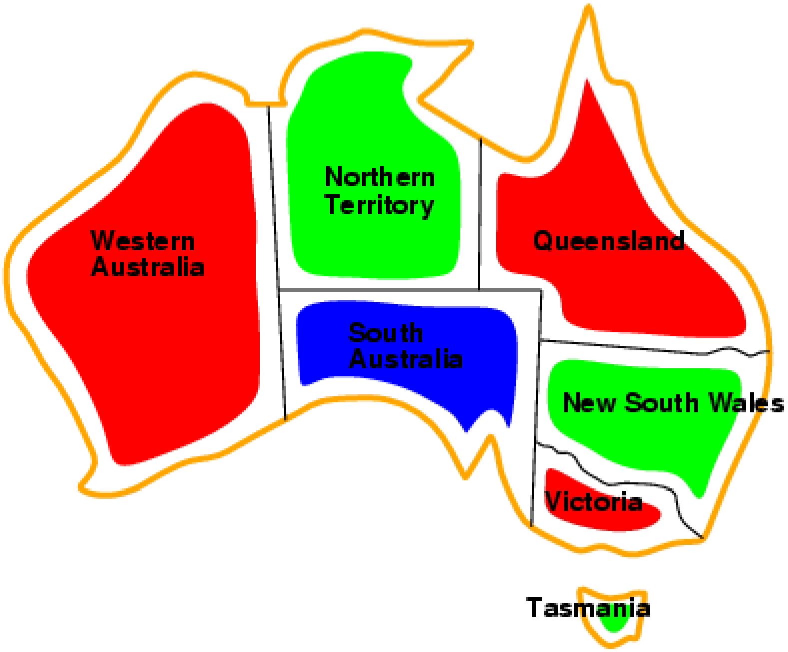
- Simple example of a formal representation language

- 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 ≠ NT, or (WA,NT) in {(red,green),(red,blue),(green,red), (green,blue),(blue,red),(blue,green)}

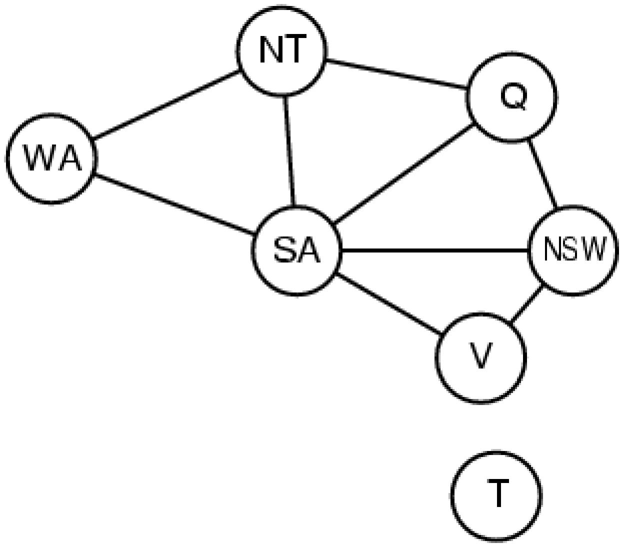
Example: Map-Coloring



- Solutions are complete and consistent assignments, e.g., WA = red, NT = green,Q = red,NSW = green,V = red,SA = blue,T = green

Constraint graph

- Binary CSP: each constraint relates two variables
- Constraint graph: nodes are variables, arcs are constraints

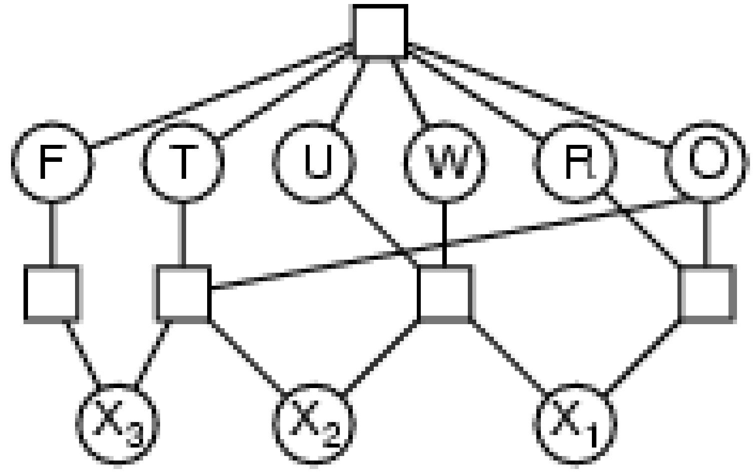


Varieties of constraints

- Unary constraints involve a single variable,
  - e.g., SA ≠ green
- Binary constraints involve pairs of variables,
  - e.g., value(SA) ≠ value(WA)
- Higher-order constraints involve 3 or more variables,
  - e.g., cryptarithmetic column constraints

Example: Cryptarithmetic

```
   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)*□
  - $O + O = R + 10 \cdot X_1$
  - $X_1 + W + W = U + 10 \cdot X_2$
  - $X_2 + T + T = O + 10 \cdot X_3$
  - $X_3 = F$, $T \neq 0$, $F \neq 0$□

Example: Latin Squares Puzzle

X1	X2	X3	X4
X5	X6	X7	X8
X9	X10	X11	X12
X13	X14	X15	X16

Variables

red	RT	RS	RC	RO
green	GT	GS	GC	GO
blue	BT	BS	BC	BO
yellow	YT	YS	YC	YO

Values

Constraints: In each row, each column, each major diagonal, there must be no two markers of the same color or same shape.

How can we formalize this?

V:

D:

C:

Real-world CSPs

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

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

The Consistent Labeling Problem

- Let P = (V,D,C) be a constraint satisfaction problem.
- An assignment is a partial function f : V -> D that assigns a value (from the appropriate domain) to each variable
- A consistent assignment or consistent labeling is an assignment f that satisfies all the constraints.
- A complete consistent labeling is a consistent labeling in which every variable has a value.

Standard Search Formulation

- state: (partial) assignment
- initial state: the empty assignment { }
- successor function: assign a value to an unassigned variable that does not conflict with current assignment
  → fail if no legal assignments
- goal test: the current assignment is complete
  (and is a consistent labeling)

1. This is the same for all CSPs regardless of application.

2. Every solution appears at depth *n* with *n* variables
   → we can use depth-first search.

3. Path is irrelevant, so we can also use complete-state formulation.

What Kinds of Algorithms are used for CSP?

- Backtracking Tree Search
- Tree Search with Forward Checking
- Tree Search with Discrete Relaxation (arc consistency, k-consistency)
- Many other variants
- Local Search using Complete State Formulation

Backtracking Tree 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.

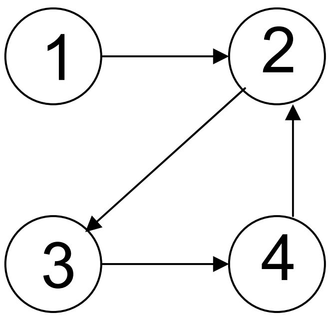
- Depth-first search for CSPs with single-variable assignments is called backtracking search.

- Backtracking search is the basic uninformed algorithm for CSPs.

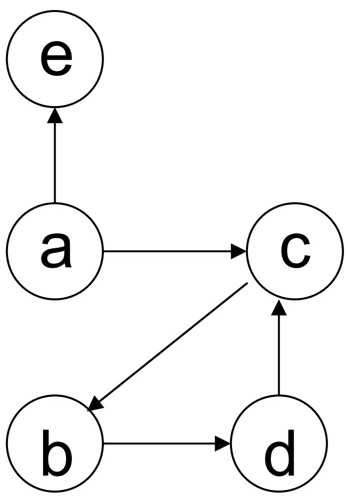
- Can solve *n*-queens for $n \approx 25$.

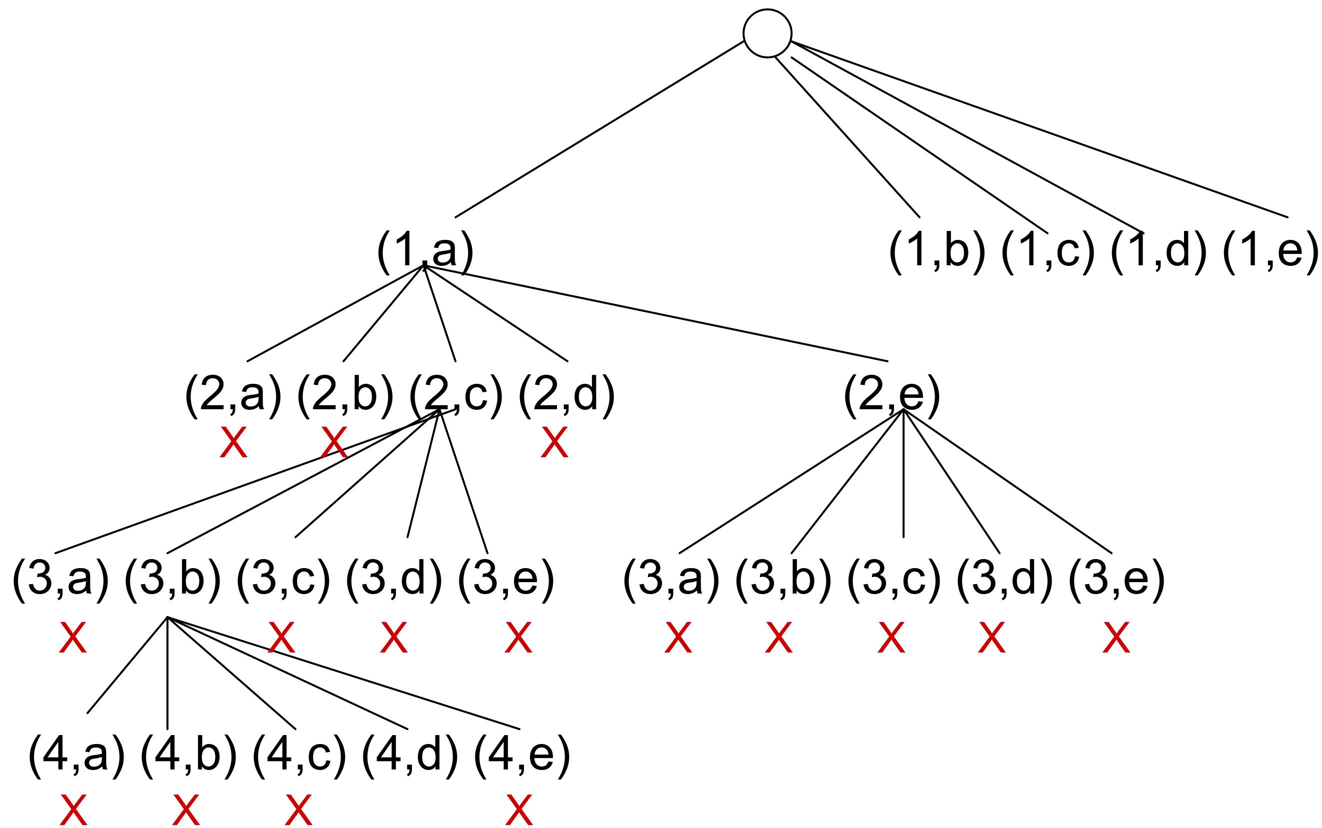
Graph Matching Example
Find a subgraph isomorphism from R to S.

R



S





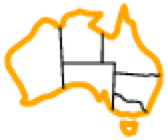
How do we formalize this problem?

Backtracking Search

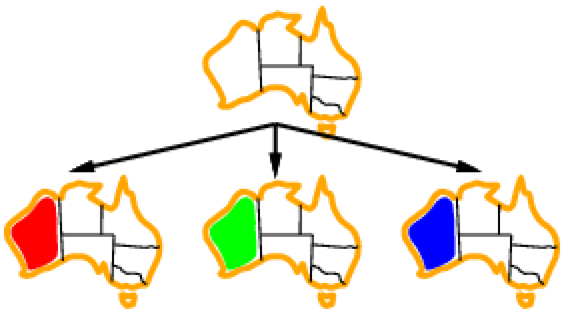
function BACKTRACKING-SEARCH(*csp*) **returns** a solution, or failure
  return RECURSIVE-BACKTRACKING({}, *csp*)

function RECURSIVE-BACKTRACKING(*assignment*, *csp*) **returns** a solution, or 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* according to Constraints[*csp*] **then**
      add { *var* = *value* } to *assignment*
      result ← RECURSIVE-BACKTRACKING(*assignment*, *csp*)
      if *result* ≠ *failue* **then return** *result*
      remove { *var* = *value* } from *assignment*
  return *failure*

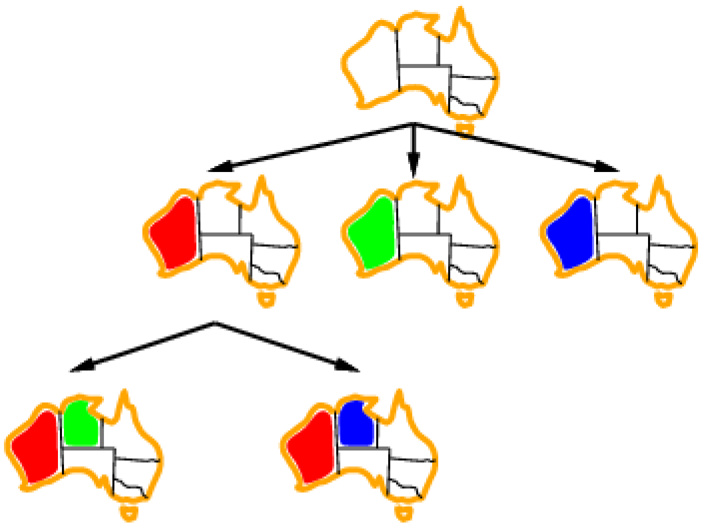
Backtracking Example



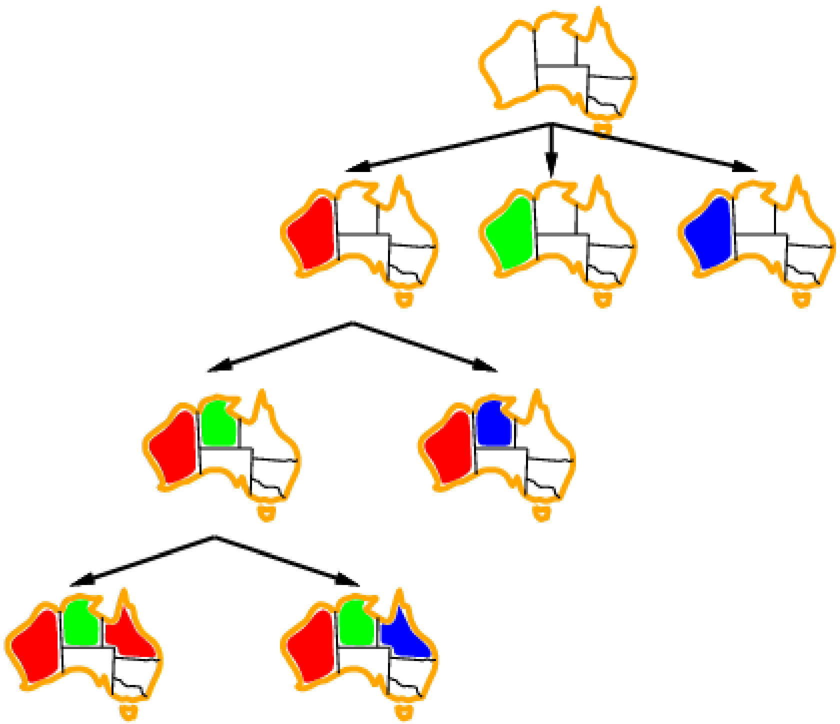
Backtracking Example



Backtracking Example



Backtracking Example



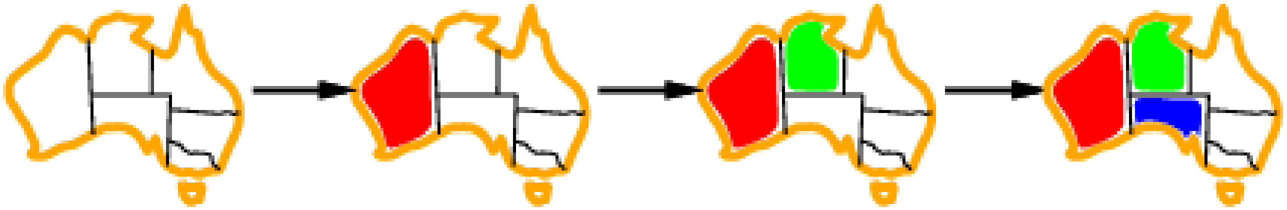
Improving Backtracking Efficiency

- General-purpose methods can give huge gains in speed:
  - Which variable should be assigned next?
  - In what order should its values be tried?
  - Can we detect inevitable failure early?

Most Constrained Variable

- Most constrained variable:

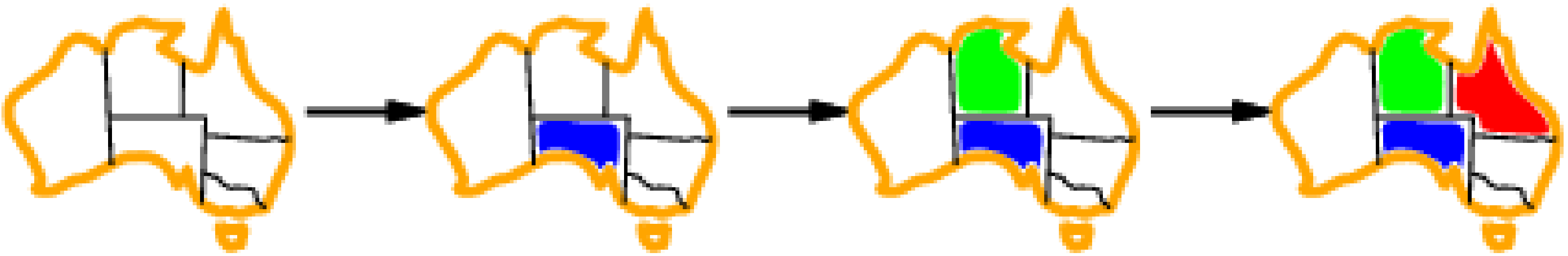
  choose the variable with the fewest legal values



- a.k.a. minimum remaining values (MRV) heuristic

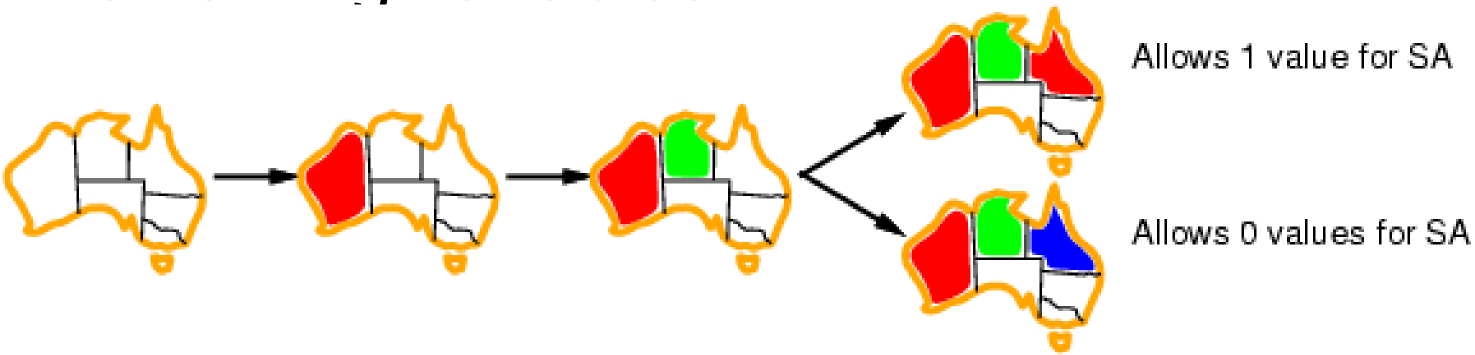
Most Constraining Variable

- Tie-breaker among most constrained variables
- Most constraining variable:
  - choose the variable with the most constraints on remaining variables



Least Constraining Value

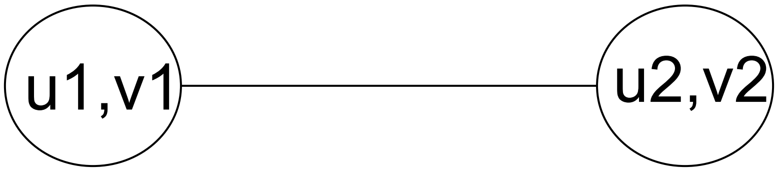
- Given a variable, choose the least constraining value:
  – the one that rules out the fewest values in the remaining variables



- Combining these heuristics makes 1000 queens feasible

Forward Checking
(Haralick and Elliott, 1980)

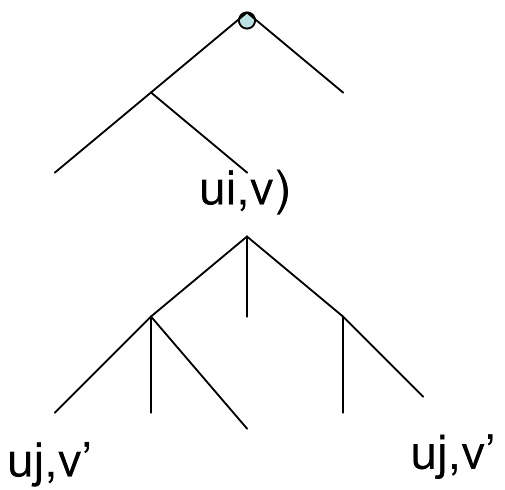
Variables: U = {u1, u2, … , un}
Values: V = {v1, v2, … , vm}
Constraint Relation: R = {(u1,v1,u2,v2) | u1 having value v1 is compatible with u2 having label v2}



If (u1,v1,u2,v2) is not in R, they are incompatible, meaning if u1 has value v1, u2 cannot have value v2.

Forward Checking

Forward checking is based on the idea that once variable ui is assigned a value v, then certain future variable-value pairs (uj,v') become impossible.



Instead of finding this out at many places on the tree, we can rule it out in advance.

Data Structure for Forward Checking

Future error table (FTAB)

One per level of the tree (ie. a stack of tables)

	v1	v2	. . .	vm
u1				
u2				
:				
un				

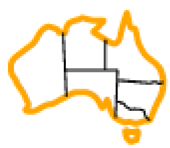
What does it mean if a whole row becomes 0?

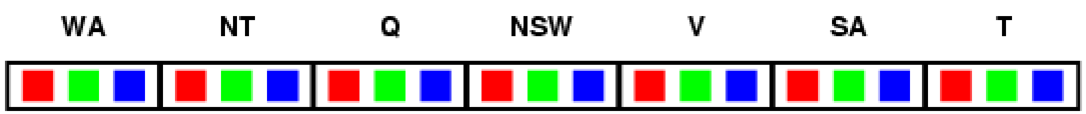
At some level in the tree,
for future (unassigned) variables u
FTAB(u,v) = 1 if it is still possible to assign v to u
0 otherwise

How do we incorporate forward checking into a backtracking depth-first search?

Book's Forward Checking Example

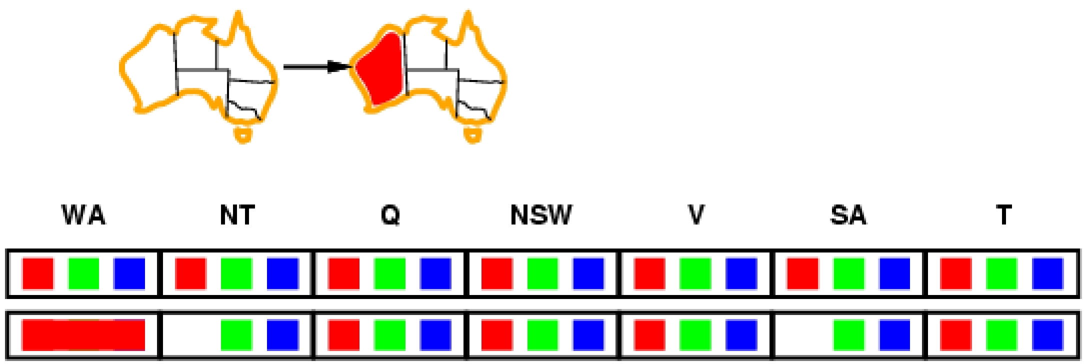
- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values





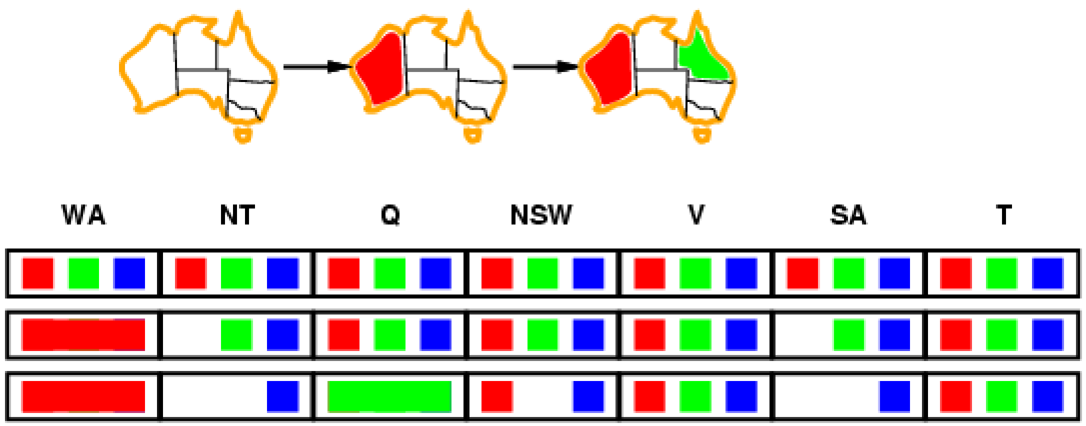
Forward Checking

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



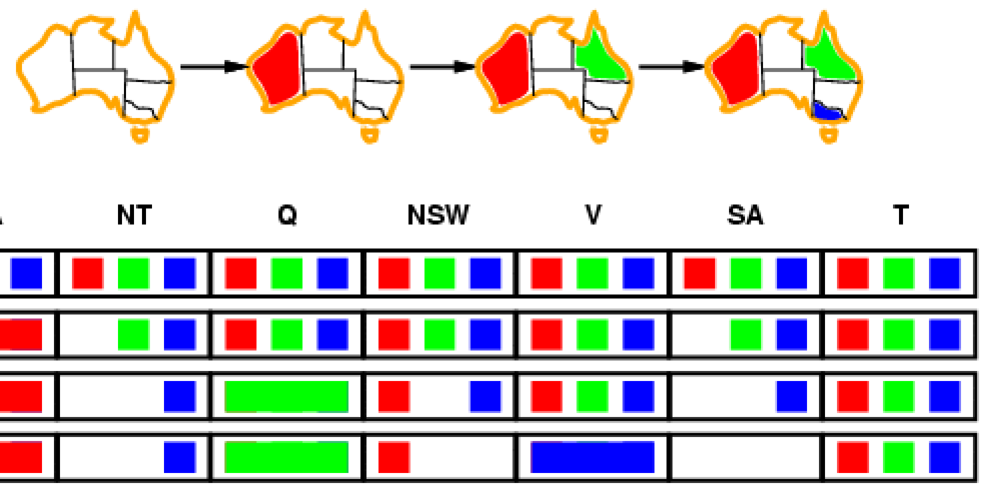
Forward Checking

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



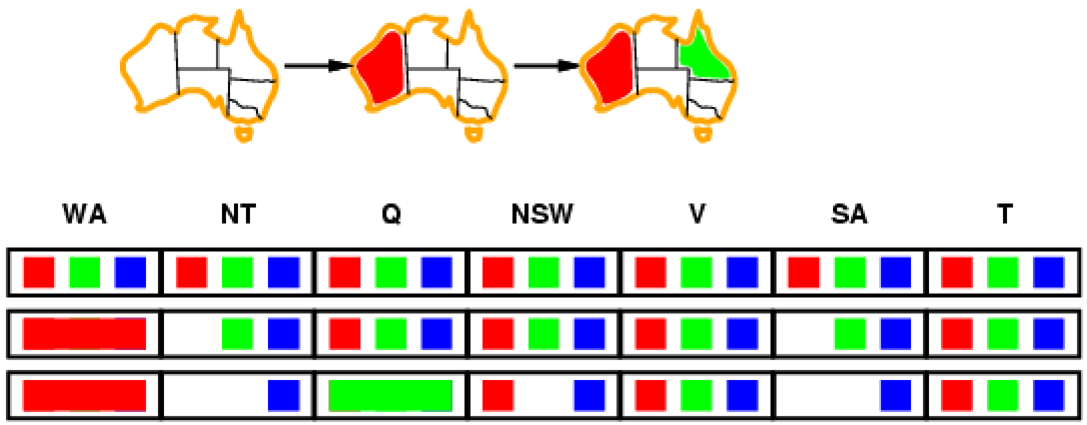
Forward Checking

- 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:

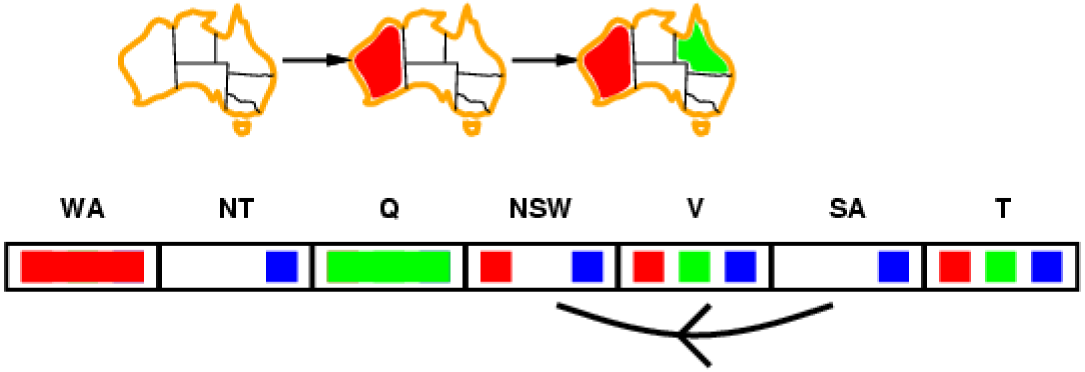


- NT and SA cannot both be blue!
- Constraint propagation repeatedly enforces constraints locally

Arc Consistency

- Simplest form of propagation makes each arc consistent
- *X* → *Y* is consistent iff

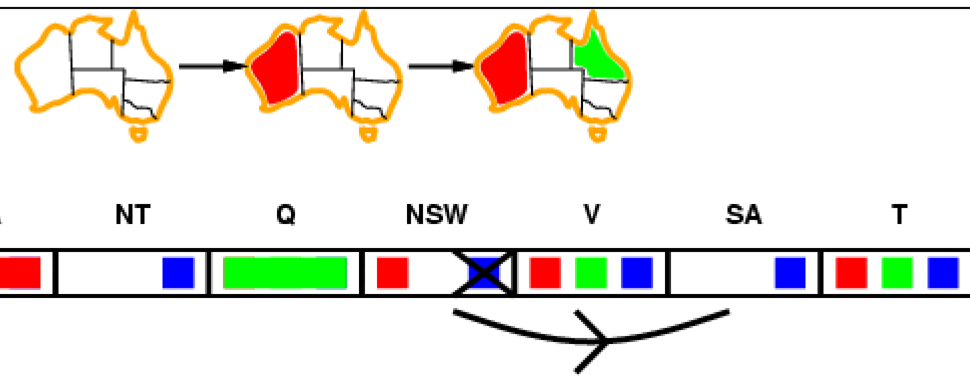
  for every value *x* of *X* there is some allowed value y of *Y*



Arc Consistency

- Simplest form of propagation makes each arc consistent
- *X* → *Y* is consistent iff

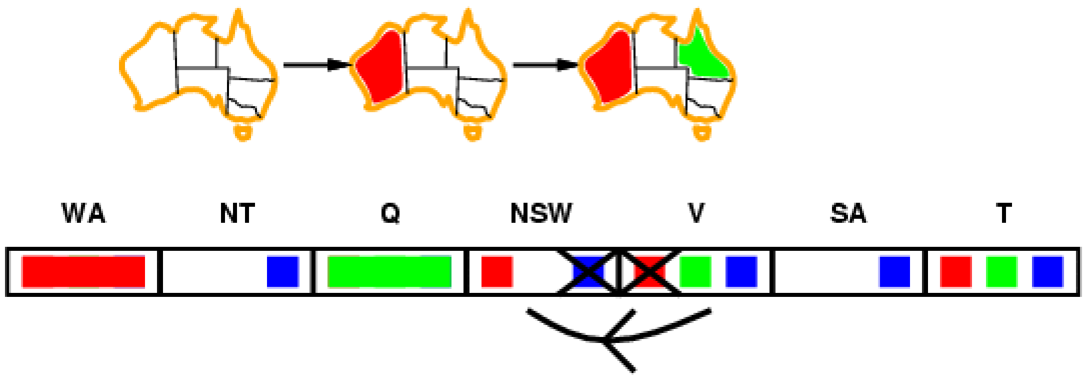
  for every value *x* of *X* there is some allowed value *y of Y*



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 value y of Y

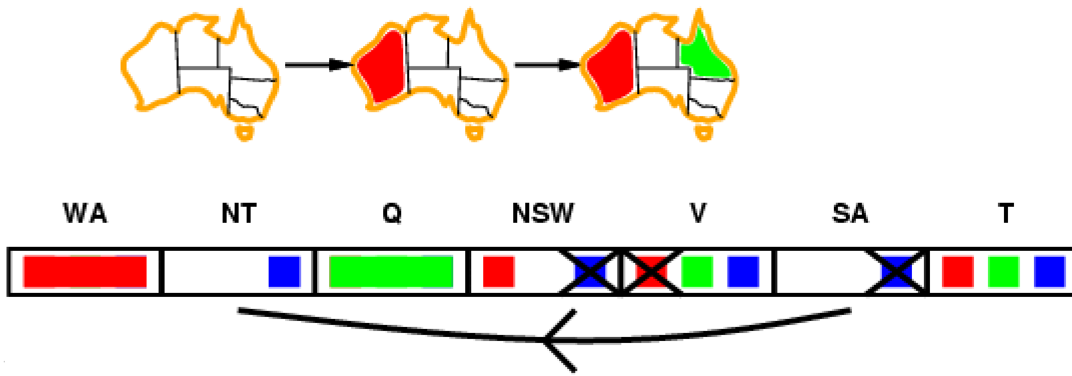


- If X loses a value, neighbors of X need to be rechecked

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 value y of Y



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

Arc Consistency Algorithm AC-3
Sometimes called Discrete Relaxation

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

  while *queue* is not empty **do**
    $(X_i, X_j) \leftarrow$ REMOVE-FIRST(*queue*)
    if RM-INCONSISTENT-VALUES(X_i, X_j) **then**
      for each X_k **in** NEIGHBORS[X_i] **do**
        add (X_k, X_i) to *queue*

---

function RM-INCONSISTENT-VALUES(X_i, X_j) **returns** true iff remove a value
  removed $\leftarrow$ *false*
  for each x **in** DOMAIN[X_i] **do**
    if no value y in DOMAIN[X_j] allows (x,y) to satisfy constraint(X_i, X_j)
      then delete x from DOMAIN[X_i]; *removed* $\leftarrow$ *true*
  return *removed*

- Time complexity: $O(n^2d^3)$□

Putting It All Together

- backtracking tree search
- with forward checking
- add arc-consistency
  - For each pair of future variables (ui,uj)
  - Check each possible remaining value v of ui
  - Is there a compatible value w of uj?
  - If not, remove v from possible values for ui (set FTAB(ui,v) to 0)

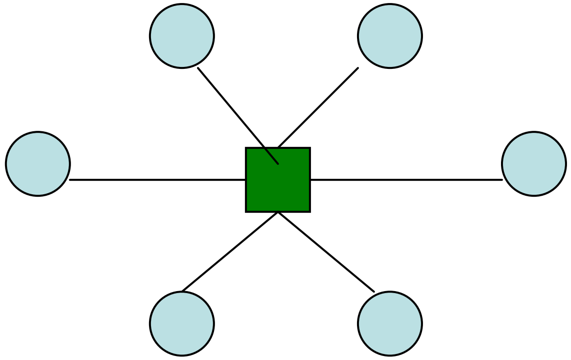
Comparison of Methods

- Backtracking tree search is a blind search.
- Forward checking checks constraints between the current variable and all future ones.
- Arc consistency then checks constraints between all pairs of future (unassigned) variables.

- What is the complexity of a backtracking tree search?
- How do forward checking and arc consistency affect that?

k-consistency (from Haralick and Shapiro, 1979, The Consistent Labeling Problem: Part I)

Variables: U = {u1, u2, … , un}
Values: V = {v1, v2, … , vm}
Constraint Relation: R = {(u1,v1,u2,v2, … uk,vk) |
u1 having value v1, u2 having value v2,…
uk having value vk are mutually compatible}



hyperarc

k-consistency

The ϕ_{kp} discrete relaxation operator tried to extend k-tuples of consistent variables and values to (k+p)-tuples of consistent variables and values in order to end up with a complete labeling consistent over all n variables and their values.

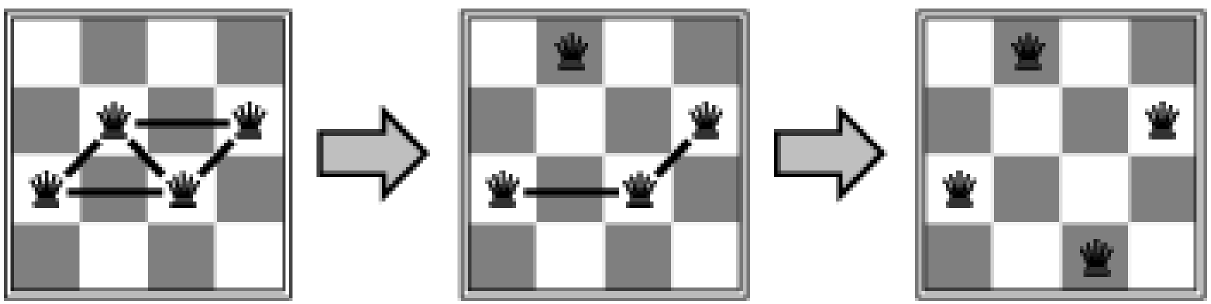
It did a great job of pruning the search, but it was very expensive to run.

Local Search for CSPs

- Hill-climbing, simulated annealing typically work with "complete" states, i.e., all variables assigned
- To apply to CSPs:
  - allow states with unsatisfied constraints
  - operators reassign variable values
- Variable selection: randomly select any conflicted variable
- Value selection by min-conflicts heuristic:
  - choose value that violates the fewest constraints
  - i.e., hill-climb with $h(n)$ = total number of violated constraints

Example: 4-Queens

- States: 4 queens in 4 columns (4^4 = 256 states)
- Actions: move queen in column
- Goal test: no attacks
- Evaluation: *h(n)* = number of attacks



h = 5 h = 2 h = 0

- Given random initial state, can solve *n*-queens in almost constant time for arbitrary *n* with high probability (e.g., *n* = 10,000,000)

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
- Iterative min-conflicts is often effective in practice