BLG435E
Artificial Intelligence

Lecture 4: Constraint Satisfaction Problems

Outline

- CSP problem formulation
- CSP examples
- Backtracking search for CSPs
- Problem structure and problem decomposition
- Local search for CSPs

Constraint Satisfaction Problems

- Search algorithms so far:
  - state is a “black box”
  - domain specific heuristics
  - states are accessible by problem specific routines
- CSP:
  - stuctured and simple representation
  - general purpose algorithms

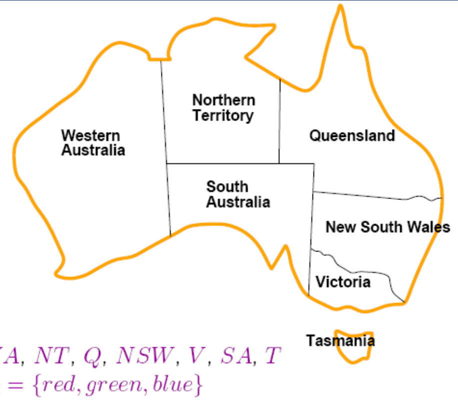
Constraint Satisfaction Problem

- Defined by
  - n variables X_i which define a state
    - Each variable has a domain D_i of possible values
  - m constraints C_j
    - Each constraint involves some subset of variables
    - Specifies the allowable combinations of values
- A state of the problem: assignment of values to some or all of X_i s

Constraint Satisfaction Problem

- Consistent or legal assignment does not violate constraints
- Complete assignment that satisfies all constraints is a solution
- A complementary objective function may be defined

Example: Map-Coloring



Variables WA, NT, Q, NSW, V, SA, T

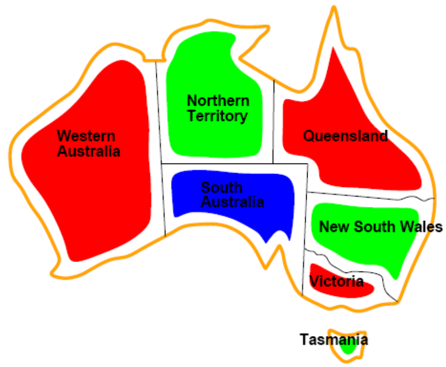
Domains $D_i = \{red, green, blue\}$

Constraints: adjacent regions must have different colors

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

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

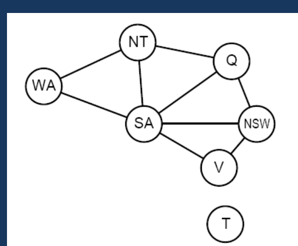
Example: Map-Coloring



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

Constraint graph

- Constraint graph: nodes are variables, arcs show constraints
- General-purpose CSP algorithms use the graph structure
  - to speed up search. e.g., Tasmania



Standard search formulation

- 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 the current assignment
    - fail if no legal assignments (not fixable!)
  - Goal test: the current assignment is complete
  - Path cost: a constant cost for every step

Standard search formulation

- The search formulation is the same for all CSPs!
- What is the depth of solution?
  - Which type of search?
- Path is irrelevant, so can also use complete-state formulation
- The number of leaves! vs possible assignments
  - Commutativity
  - Consider only a single variable at each node

Varieties of CSPs

- Discrete variables
  - finite domains; size d → $O(d^n)$ complete assignments
    - e.g., Boolean CSPs including 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., $StartJob1 + 5 \leq StartJob3$
    - linear constraints solvable, nonlinear undecidable

Varieties of CSPs

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

Varieties of constraints

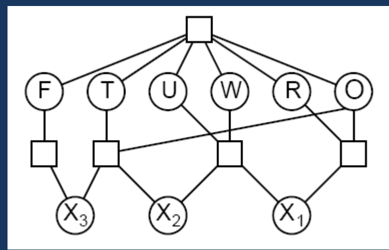
- Unary constraints involve a single variable
  - SA $\neq$ green
- Binary constraints involve pairs of variables
  - SA $\neq$ WA
- A binary CSP has only binary constraints, constraint graphs

Varieties of constraints

- Global constraints involve an arbitrary number of variables: cryptarithmetic column constraints
  - constraint hypergraph
  - can be reduced to binary constraints
- Preferences (soft constraints)
  - red is better than green
  - often encoded using costs againts the overall objective function
  - constrained optimization problems

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)

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

Real-world CSPs

- Assignment problems
  - who teaches what class
- Timetabling problems
  - which class is offered when and where?
- Hardware configuration
- Spreadsheets
- Transportation scheduling
- Factory scheduling
- Floorplanning

Benefits of modeling as a CSP

- Representation of states conforms to a standard pattern
- The successor function and goal test can be written in a generic way
- Devising generic heuristics
- The structure of the constraint graph can be used to simplify the solution process

Backtracking search

- Variable assignments are commutative
  - [WA=red then NT =green] same as [NT =green then WA=red]
- Only need to consider assignments to a single variable at each node
  - b=d and there are d^n leaves

Backtracking search

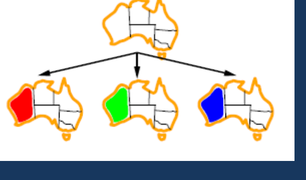
- Depth-first search with single-variable assignments
- The basic uninformed algorithm for CSPs
- Can solve n-queens for n $\approx$ 25

Backtracking search

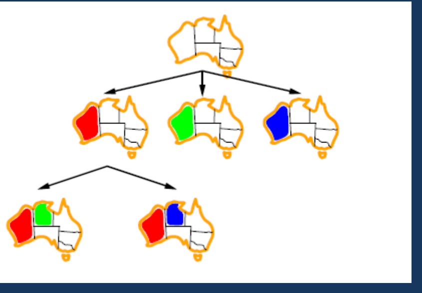
```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
  return BACKTRACK({ }, csp)

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 example

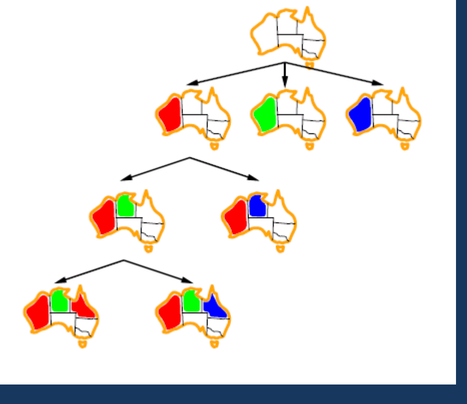
Backtracking example



Backtracking example



Backtracking example

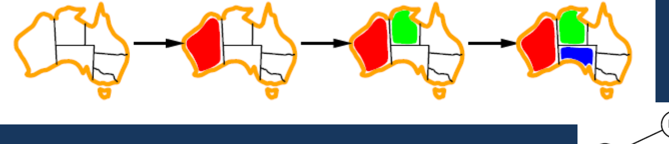


Improving backtracking efficiency

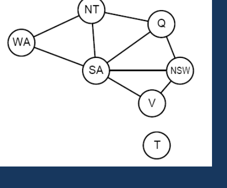
- General-purpose methods can give huge gains in speed:
  - Which variable should be assigned next?
  - In what order should its values be tried?
  - Can we detect inevitable failure early?
  - Can we take advantage of problem structure?

Minimum remaining values

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

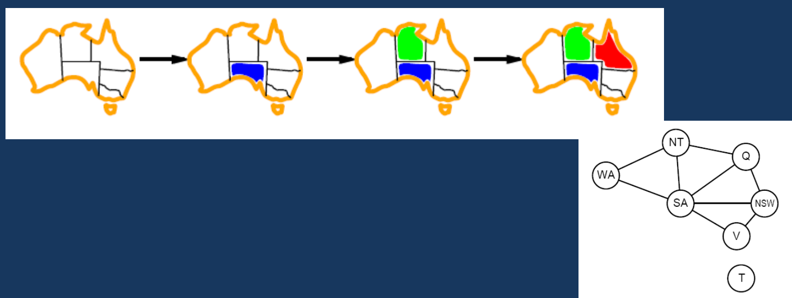


  - Most constrained variable, fail-first heuristic



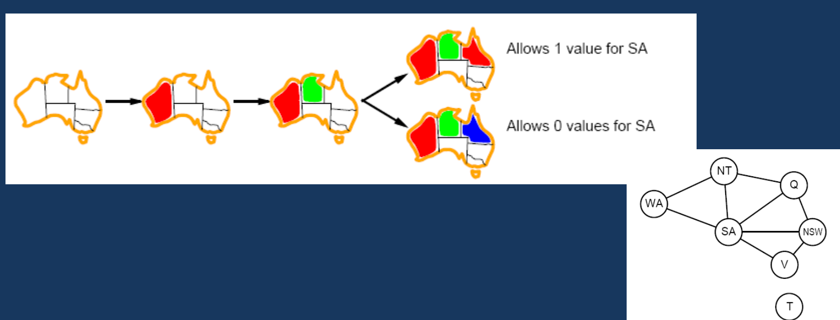
Degree heuristic

- Tie-breaker among MRV variables
- Degree heuristic:
  - choose the variable with the most constraints on remaining variables



Least constraining value

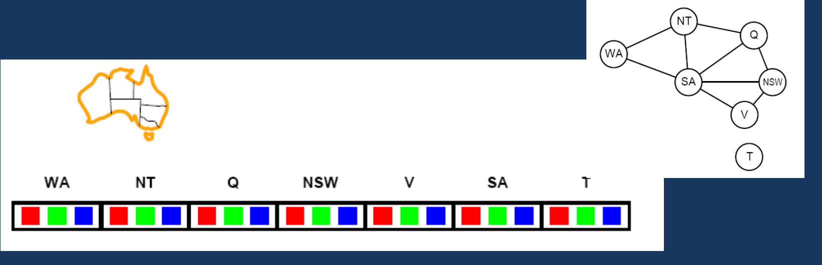
- Given a variable, choose the least constraining value:
  - the one that rules out the fewest values in the remaining variables



- Combining these heuristics makes 1000 queens feasible

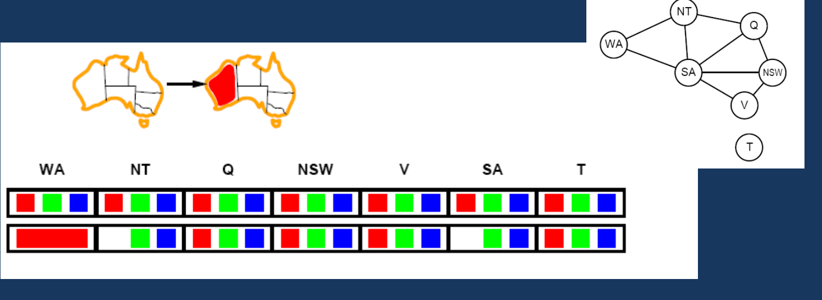
Forward checking

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



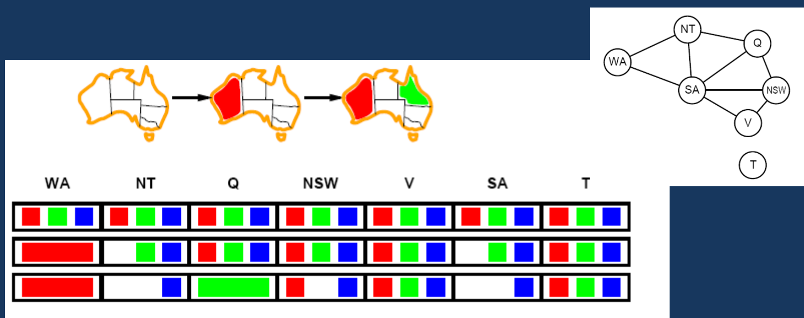
Forward checking

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



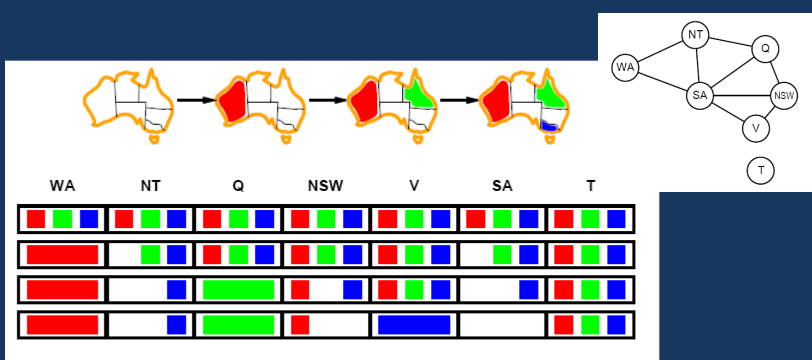
Forward checking

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



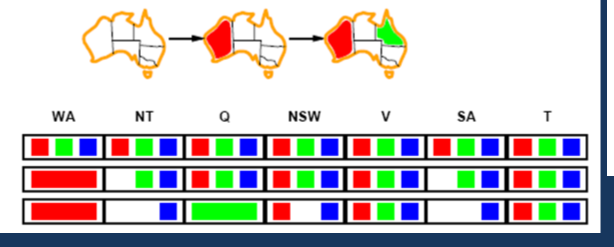
Forward checking

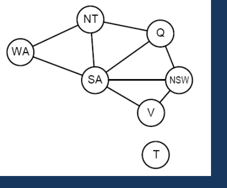
- Idea: Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values



Constraint propagation

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



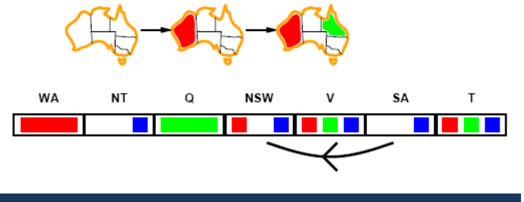


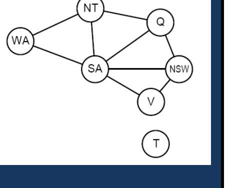
Arc consistency

- Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent iff

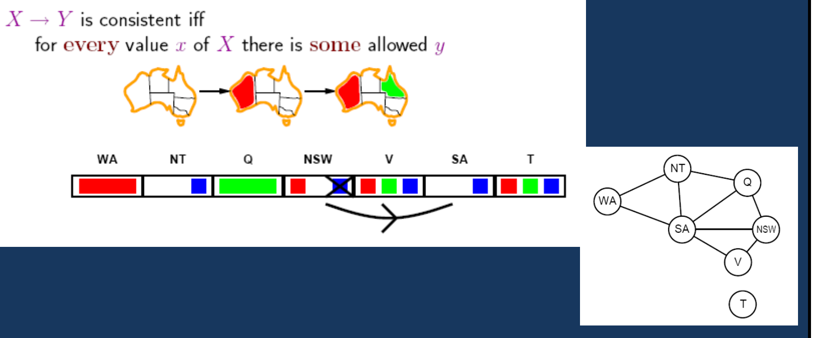
for every value x of X there is some allowed y





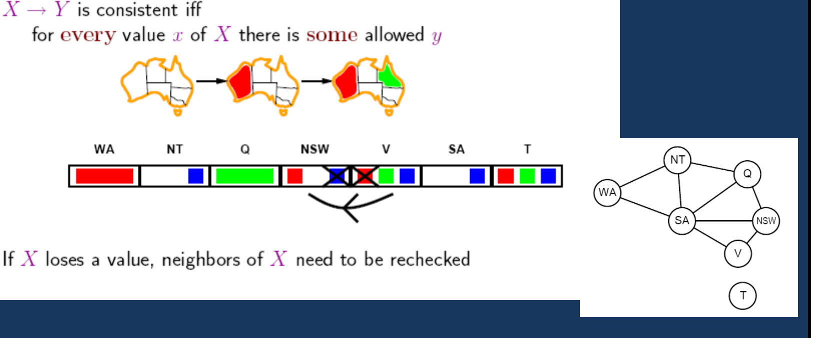
Arc consistency

- Simplest form of propagation makes each arc consistent



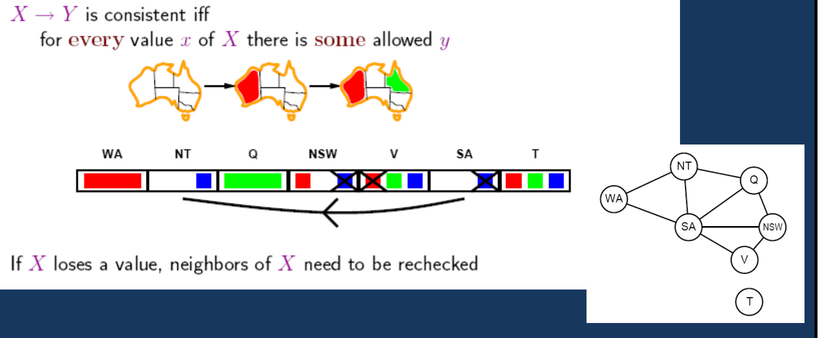
Arc consistency

- Simplest form of propagation makes each arc consistent



Arc consistency

- Simplest form of propagation makes each arc consistent



Arc consistency algorithm

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*

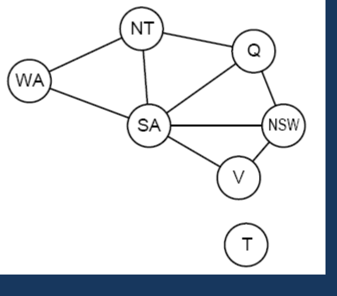
---

function REVISE(*csp*, X_i, X_j) **returns** true iff we revise the domain of X_i
  revised ← *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 ← *true*
  return *revised*

k-consistency

- Arc-consistency does not reveal every possible inconsistency
- k-consistency:
  - for any k-1 variables and
  - for any consistent assignment
  - A consistent value can be assigned to any k^{th} variable
- 1-consistency: node consistency
- 2-consistency: arc consistency
- 3-consistency: path consistency
- A graph is strongly k-consistent if it is k-consistent and also (k-1) –consistent, ... 1-consistent

Problem structure



Problem structure

Suppose each subproblem has c variables out of n total

Worst-case solution cost is $n/c \cdot d^c$, **linear** in n

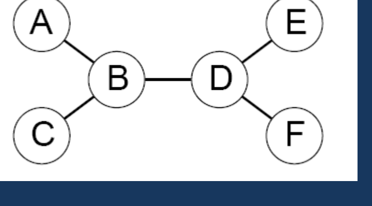
E.g., $n = 80$, $d = 2$, $c = 20$

2^{80} = 4 billion years at 10 million nodes/sec

$4 \cdot 2^{20}$ = 0.4 seconds at 10 million nodes/sec

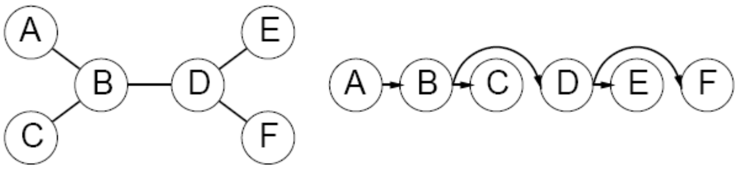
Tree-structured CSPs

- Theorem: if the constraint graph has no loops, the CSP can be solved in $O(nd^2)$



Algorithm for tree-structured CSPs

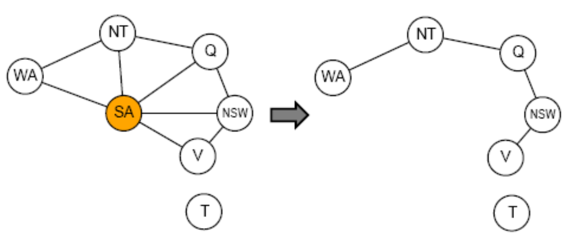
1. Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering



2. For j from n down to 2, apply $\text{RemoveInconsistent}(Parent(X_j), X_j)$
3. For j from 1 to n, assign X_j consistently with $Parent(X_j)$

Nearly tree-structured CSPs

Conditioning: instantiate a variable, prune its neighbors' domains



Cutset conditioning: instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree

Cutset size c $\Rightarrow$ runtime $O(d^c \cdot (n-c)d^2)$, very fast for small c