CSC520 - Artificial Intelligence
Lecture 10

Dr. Scott N. Gerard

North Carolina State University

February 4, 2025

Agenda

- Backtracking search
- Heuristics for improving backtracking search
- Constraint Propagation: arc consistency

Backtracking Search

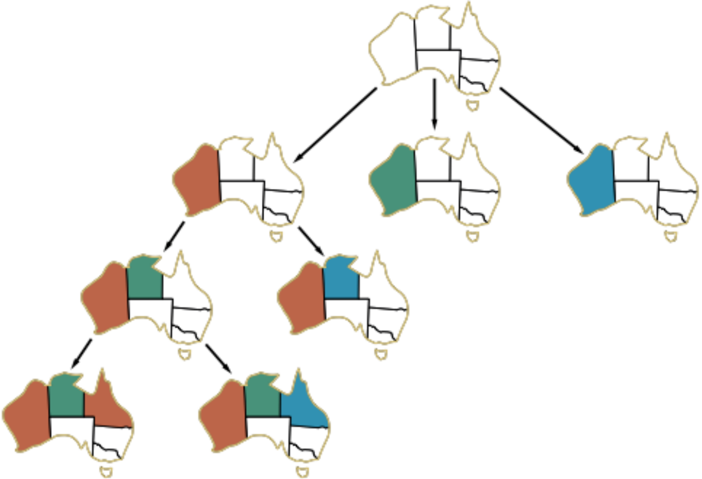
- Assign one variable at each step
  - Variable assignments are commutative
  - E.g. $WA = red$ then $NT = green$ is same as $NT = green$ then $WA = red$
  - Consider assignments to a single variable at each step
- Check constraints at each step
  - Consider values that do not conflict previous assignments
  - Will need some computation to check the constraints
- Depth first search with above improvements is called as backtracking search for CSP
- Can solve n-queen problems for $n \approx 25$

Backtracking Search

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

function RECURSIVE-BACKTRACKING(*assignment*, *csp*) **return** solution or failure
    if *assignment* is complete **then return** *assignment*
    var $\leftarrow$ SELECT-UNASSIGNED-VARIABLE(VARIABLES[*csp*], *assignment*, *csp*)
    for *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 $\leftarrow$ RECURSIVE-BACKTRACKING(*assignment*, *csp*)
            if $result \neq failure$ **then return** *result*
            remove $\{var = value\}$ from *assignment*
    return *failure*

Backtracking Search



Heuristics for Improving Backtracking Search

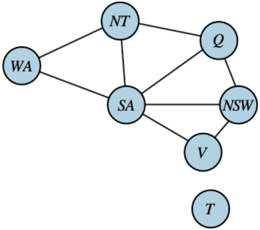
- Ordering
  - Which variable should be assigned next?
  - In what order should its values be tried?
- Forward checking
  - Can we detect inevitable failures early?

Minimum Remaining Values (MRV)

- Choose a variable with the fewest legal values

Degree Heuristic

- If multiple variables have the same MRV, choose a variable with most constraints on remaining variables



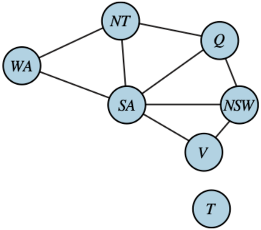
Least Constraining Values

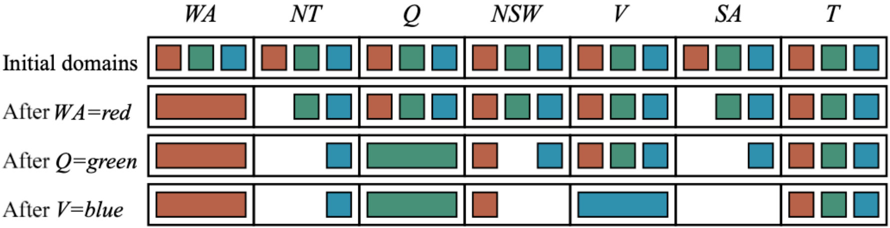
- Given a variable, choose the least constraining values

Combining these heuristics makes 1000 queens feasible

Forward Checking

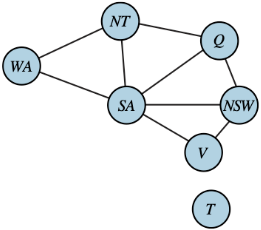
- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values

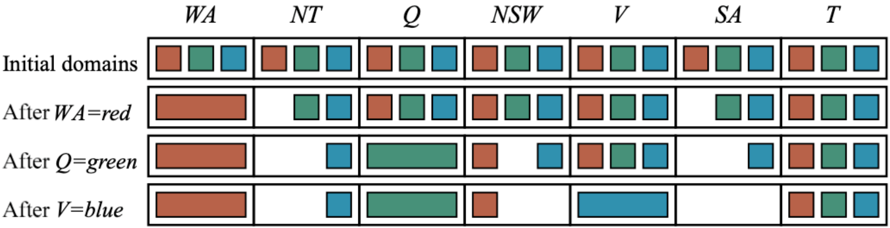




Constraint Propagation

- Forward checking propagates information from assigned to unassigned variables
- Does not provide early detection for all failures

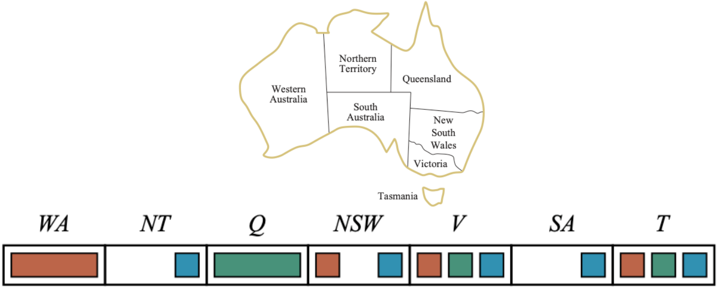




- Constraint propagation repeatedly enforces constraints locally

Arc Consistency

- A form of constraint propagation that makes each arc (binary constraint) consistent
- $X \rightarrow Y$ is consistent iff for every value $x \in X$, there is some value $y \in Y$ that satisfies the constraint



Is $SA \rightarrow NSW$ arc consistent?

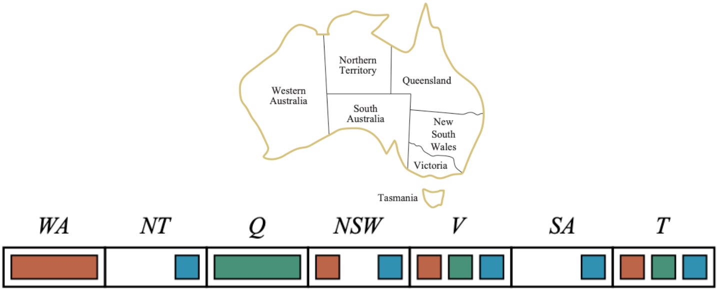
Arc Consistency

- A form of constraint propagation that makes each arc (binary constraint) consistent
- $X \rightarrow Y$ is consistent iff for every value $x \in X$, there is some value $y \in Y$ that satisfies the constraint

Is $NSW \rightarrow SA$ arc consistent?

Arc Consistency

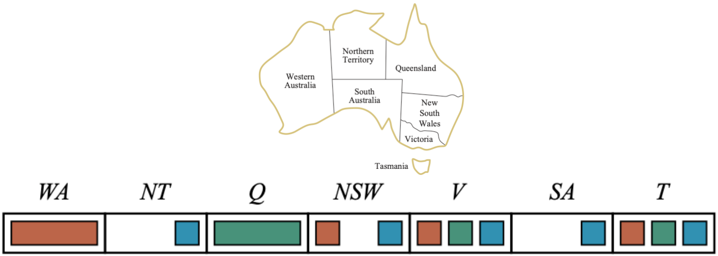
- A form of constraint propagation that makes each arc (binary constraint) consistent
- $X \rightarrow Y$ is consistent iff for every value $x \in X$, there is some value $y \in Y$ that satisfies the constraint



- If X loses a value, then neighbors of X need to be rechecked
- E.g. $V \rightarrow NSW$

Arc Consistency

- A form of constraint propagation that makes each arc (binary constraint) consistent
- $X \rightarrow Y$ is consistent iff for every value $x \in X$, there is some value $y \in Y$ that satisfies the constraint



- Arc consistency detects failures earlier than forward checking
- E.g. $SA \rightarrow NT$
- Can be run as a preprocessor step or after assignment during search

Arc Consistency Algorithm (AC3)

function $\mathrm{AC3}(csp)$ **return** false or true

queue $\leftarrow$ initially all arcs in *csp*

while *queue* is not empty **do**

$(X_i, X_j) \leftarrow \textsc{pop}(queue)$

if $\textsc{revise}(csp, X_i, X_j)$ **then**

if size of $D_i = 0$ **then return** *false*

for X_k in $\textsc{neighbors}(X_i)$ **do**

add (X_k, X_i) to *queue*

return *true*

function $\textsc{revise}(csp, X_i, X_j)$ **return** false or true

revised $\leftarrow$ *false*

for x in D_i **do**

if no value in D_j allows (x, y) to satisfy $X_i \leftrightarrow X_j$ constraint **then**

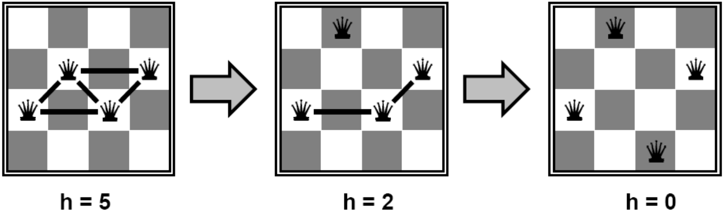
delete x from D_i; *revised* $\leftarrow$ *true*

return *revised*

Local Search for CSPs

- Local search algorithms such as hill climbing or simulated annealing can be very effective in solving many CSPs
- To apply local search to CSP
  - Initial state has random assignment of variables with possible unsatisfied constraints
  - Randomly select any conflicted variable
  - Min-conflicts heuristic: Choose a value for the variable that violates fewest constraints
  - Repeat until no conflicts
- Very efficient in solving certain CSPs
- Can solve n-queens up to $n = 1,000,000$ in an average of 50 steps

Example 4-Queens



- $\mathcal{X} = \{Q_1, Q_2, Q_3, Q_4\}$
- $D_i = \{1, 2, 3, 4\}$
- Constraints: $\forall i, j$ NonAttacking(Q_i, Q_j) OR $(Q_1, Q_2) \in \{(1,3), (1,4), \ldots\}$

- Initial state: $Q_1 = 2, Q_2 = 3, \ldots$
- $h =$ Number of conflicts

Class Exercise

Consider the following CSP.

- Variables: X, Y, Z
- Domains: $D_X = \{1, \ldots 10\}, D_Y = \{5, \ldots 15\}, D_Z = \{5, \ldots 20\}$
- Constraints: $X > Y$, $X + Z = 16$

1. Draw the constraint graph for the CSP
2. Are the constraints arc consistent? If not, apply arc consistency method repeatedly so they become arc consistent. What is the updated domain of each variable?