CSC520 - Artificial Intelligence

Lecture 9

Dr. Scott N. Gerard

North Carolina State University

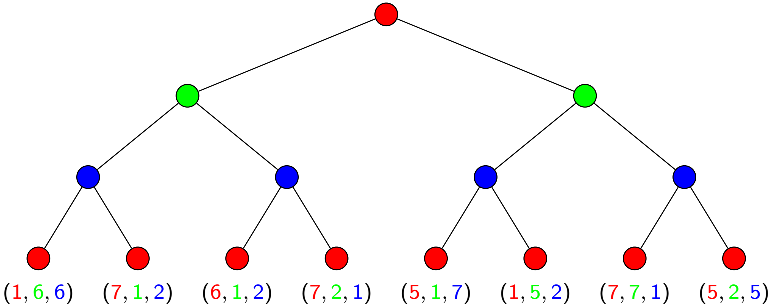
Feb 4, 2025

Agenda

- Multiplayer Games
- Constraint Satisfaction Problem (CSP)

Multiplayer Games

- States have utility tuples
- Each player maximizes its own utility value. No longer zero-sum
- Gives rise to cooperation and competition
- Generalized minimax search



Constraint Satisfaction Problem

- Factored representation of state
- Three components of CSP: $\mathcal{X}, \mathcal{D}, \mathcal{C}$
  - (X) is a set of variables: $\{X_1, X_2, \ldots, X_n\}$
  - (D) is a set of domains: $\{D_1, D_2, \ldots, D_n\}$
  - (C) is a set of constraints
- A solution is an assignment of value to each variable that satisfies all constraints

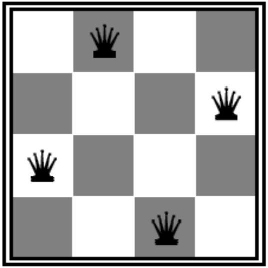
Map Coloring Problem

- $\mathcal{X} = \{WA, NT, Q, NSW, V, SA, T\}$
- $D_i = \{red, green, blue\}$
- $\mathcal{C} = \{SA \neq WA, SA \neq NT, SA \neq Q, \ldots\}$

Solution: $\{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green\}$

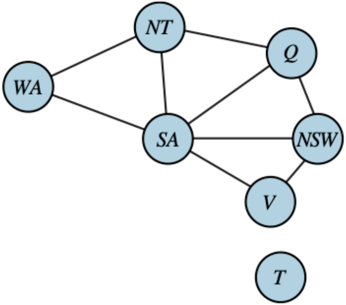
N-Queens Problem

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



Constraint Graph

- Nodes are variables
- Edges represent constraints among variables
- Binary CSP: each constraint relates at most two variables



Variables and Constraints

- Discrete variables
  - Finite domain, e.g. $D = \{1, 2, 3\}$
    - Size d for n variables means $O(d^n)$ assignments
    - E.g., Boolean CSPs including Boolean satisfiability (NP-complete)
  - Infinite domain, e.g. $D =$ Set of all integers
    - E.g. job scheduling where variables are start or end times of each task
- Continuous variables
  - E.g. scheduling experiments on Hubble Telescope where start and end of observations are continuous variables
  - Linear constraints solvable in polynomial time by Linear Programming methods

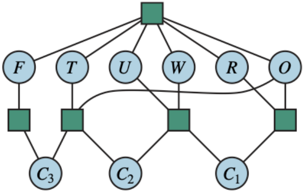
Types of Constraints

- Hard constraints
  - Unary constraint involve a single variable, e.g. $SA = green$
  - Binary constraint involves a pair of variables, e.g. $SA \neq WA$
  - Higher-order constraint involves 3 or more variables
    - *Alldiff* constraint means that all variables must have different values
- Preference constraints are soft constraints, e.g. red is better than green
  - Can be encoded as costs on variable assignments
  - CSPs with preferences are constrained optimization problems which can be solved using Linear Programming methods

Cryptarithmetic

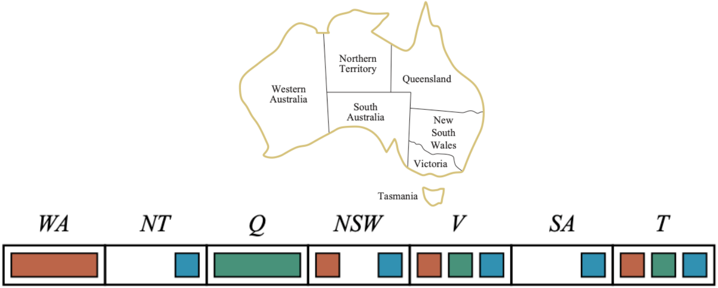
- $\mathcal{X} = \{F, T, U, W, R, O, C_1, C_2, C_3\}$
- $D_i = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Constraints
  - $Alldiff(F, T, U, W, R, O)$
  - $O + O = R + 10 \cdot C_1$
  - $C_1 + W + W = U + 10 \cdot C_2$
  - $C_2 + T + T = O + 10 \cdot C_3$
  - $C_3 = F$

$$\begin{array}{ccccc} & & T & W & O \\ + & & T & W & O \\ \hline & F & O & U & R \end{array}$$



Arc Consistency

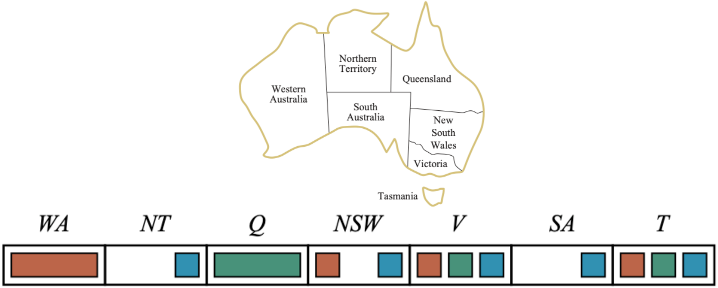
- A form of constraint propagation that makes each arc (binary constraint) consistent
- $X \rightarrow Y$ is consistent iff for every value $x \in X$, there is some value $y \in Y$ that satisfies the constraint



Is $SA \rightarrow NSW$ arc consistent?

Arc Consistency

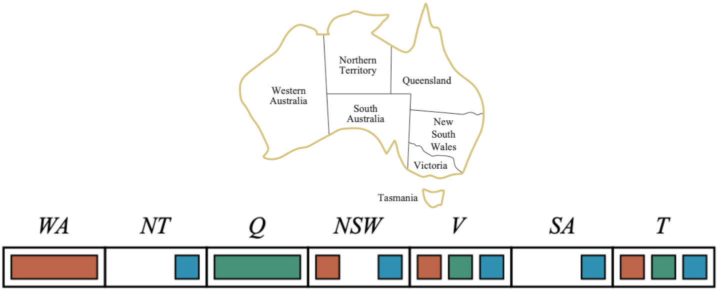
- A form of constraint propagation that makes each arc (binary constraint) consistent
- $X \rightarrow Y$ is consistent iff for every value $x \in X$, there is some value $y \in Y$ that satisfies the constraint



Is $NSW \rightarrow SA$ arc consistent?

Arc Consistency

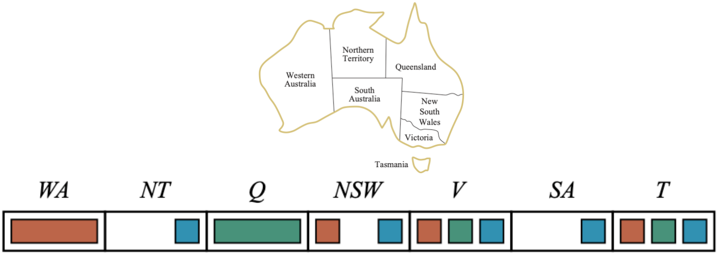
- A form of constraint propagation that makes each arc (binary constraint) consistent
- $X \rightarrow Y$ is consistent iff for every value $x \in X$, there is some value $y \in Y$ that satisfies the constraint



- 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*

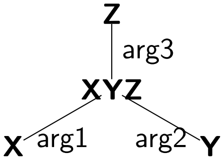
  return *revised*

Arc Consistency Algorithm (AC3)

- AC-3 only works for binary constraints.
- We can always convert n-ary constraints to binary constraints.
- Convert $X + Y = Z$. Domains are [0, 1, 2, . . . , 9]

Arc Consistency Algorithm (AC3)

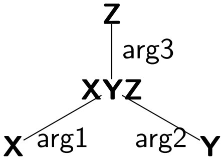
- AC-3 only works for binary constraints.
- We can always convert n-ary constraints to binary constraints.
- Convert X + Y = Z. Domains are [0, 1, 2, . . . , 9]



(X,Y,Z)	(X,Y,Z)	(X,Y,Z)	. . .	(X,Y,Z)
(0,0,0)	(0,1,1)	(0,2,2)	. . .	(0,9,9)
	(1,0,1)	(1,1,2)	. . .	(1,8,9)
		(2,0,2)	. . .	(2,7,9)
				. . .
				(9,0,9)

Arc Consistency Algorithm (AC3)

- AC-3 only works for binary constraints.
- We can always convert n-ary constraints to binary constraints.
- Convert X + Y = Z. Domains are [0, 1, 2, . . . , 9]



(X,Y,Z)	(X,Y,Z)	(X,Y,Z)	. . .	(X,Y,Z)
(0,0,0)	(0,1,1)	(0,2,2)	. . .	(0,9,9)
	(1,0,1)	(1,1,2)	. . .	(1,8,9)
		(2,0,2)	. . .	(2,7,9)
				. . .
				(9,0,9)

$n(n+1)/2 = 55$ out of $1000 = 5.5\%$

Solving CSPs - Standard Search Formulation

- States are defined by the values assigned so far
- Initial state: no variables are assigned, $\{\}$
- Successor function: assign a value to an unassigned variable
- Goal state: assignment is complete and satisfies all constraints

Solving CSPs - Standard Search Formulation

- States are defined by the values assigned so far
- Initial state: no variables are assigned, $\{\}$
- Successor function: assign a value to an unassigned variable
- Goal state: assignment is complete and satisfies all constraints
- Solution appears at depth n where n is number of variables

Backtracking Search

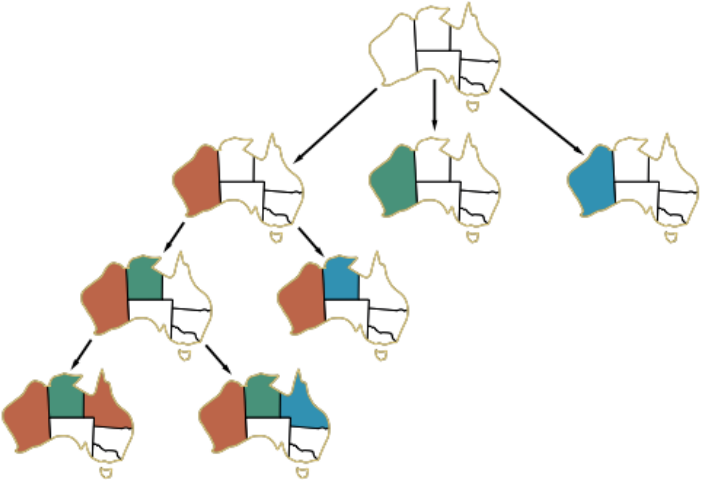
- Assign one variable at each step
  - Variable assignments are commutative
  - E.g. $WA = red$ then $NT = green$ is same as $NT = green$ then $WA = red$
  - Consider assignments to a single variable at each step
- Check constraints at each step
  - Consider values that do not conflict previous assignments
  - 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 ← 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 ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failure then return result
            remove {var = value} from assignment
    return failure
```

Backtracking Search



Improve Backtracking Search

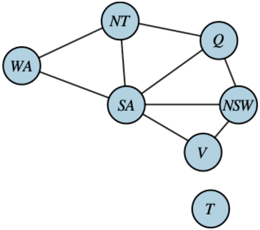
- Ordering
  - Which variable should be assigned next?
  - In what order should its values be tried?
- Forward checking
  - Can we detect inevitable failures early?
- Structure
  - Can we exploit the problem structure?

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

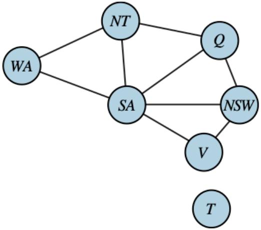


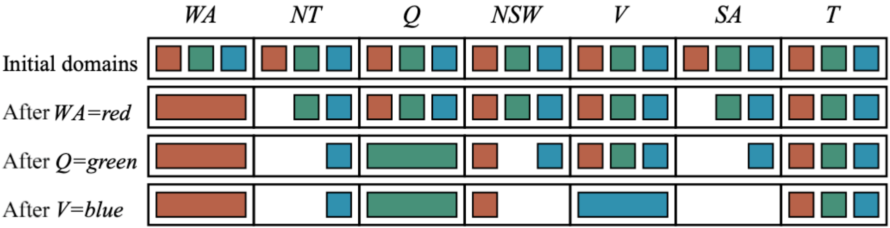
Least Constraining Values

- Given a variable, choose the least constraining values

Forward Checking

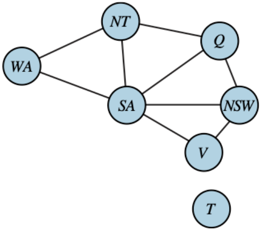
- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values

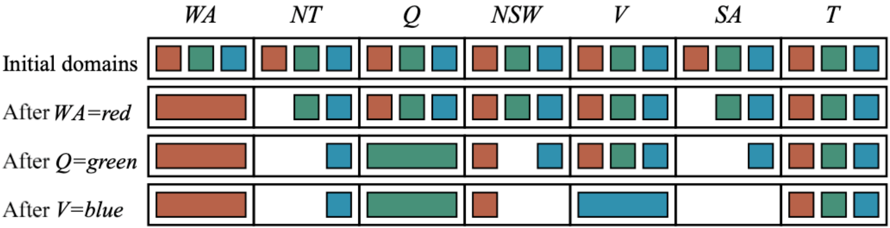




Constraint Propagation

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





- Constraint propagation repeatedly enforces constraints locally