Introduction to Artificial Intelligence and Problem Formulations

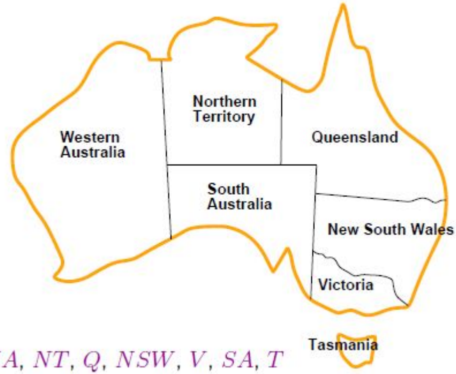
Dr. Tapas Kumar Mishra

Assistant Professor
Department of Computer Science and Engineering
SRM University
Amaravati-522502, Andhra Pradesh, India
Email: tapaskumar.m [at] srmap [dot] edu [dot] com

Constraint Satisfaction Problems (CSP)

- A CSP is defined by a set of **variables**, X_1, X_2,..., X_n and a set of **constraints**, C_1, C_2,..., C_m.
- Each variable X_i has a nonempty domain D_i of possible values.
- A state of the problem is defined by an **assignment** of values to some or all of the variables, $\{X_i = v_i, X_j = v_j, \ldots\}$.
- An assignment that does not violate any constraints is called a **consistent** or legal assignment.

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

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

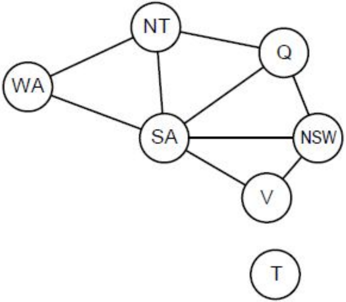
Example: Map-Coloring contd.

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

Constraint graph

Binary CSP: each constraint relates at most two variables

Constraint graph: nodes are variables, arcs show constraints



General-purpose CSP algorithms use the graph structure
to speed up search. E.g., Tasmania is an independent subproblem!

Varieties of CSPs

Discrete variables

finite domains; size $d \Rightarrow O(d^n)$ complete assignments

- ◊ e.g., Boolean CSPs, incl. 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., $StartJob_1 + 5 \leq StartJob_3$
- ◊ linear constraints solvable, nonlinear undecidable

Continuous variables

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

Varieties of constraints

Unary constraints involve a single variable,
e.g., $SA \neq green$

Binary constraints involve pairs of variables,
e.g., $SA \neq WA$

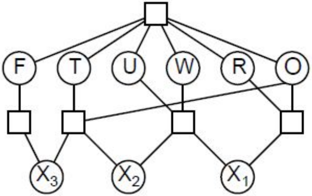
Higher-order constraints involve 3 or more variables,
e.g., cryptarithmetic column constraints

Preferences (soft constraints), e.g., *red* is better than *green*
often representable by a cost for each variable assignment
→ constrained optimization problems

Constraint Satisfaction Problems (CSP)

Example: Cryptarithmetic

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



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.

- $X_1 + W + W = U + 10 \times X_2$
- $X_2 + T + T = O + 10 \times X_3$
- $X_3 = F$

Constraint Satisfaction Problems (CSP)

Real-world CSPs

Assignment problems
e.g., who teaches what class

Timetabling problems
e.g., which class is offered when and where?

Hardware configuration

Spreadsheets

Transportation scheduling

Factory scheduling

Floorplanning

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

- Prof. *X* might prefer teaching in the morning whereas Prof. *Y* prefers teaching in the afternoon.
- A timetable that has Prof. *X* teaching at 2 p.m. would still be a **solution**, but would **not** be an **optimal** one.

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

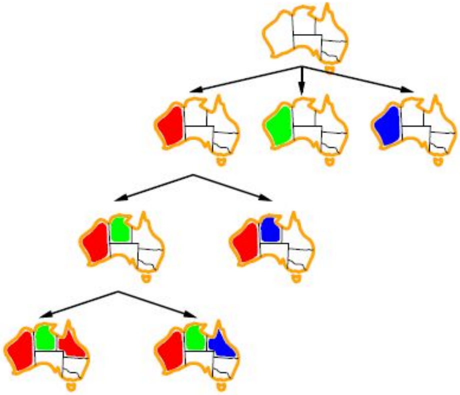
Depth-first search for CSPs with single-variable assignments
is called backtracking search

Backtracking search is the basic uninformed algorithm for CSPs

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

Backtracking Search for CSPs

Backtracking example



Backtracking Search for CSPs

Improving backtracking efficiency

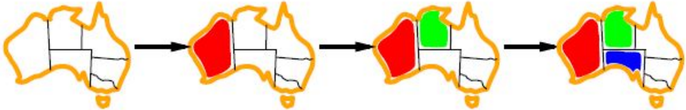
General-purpose methods can give huge gains in speed:

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

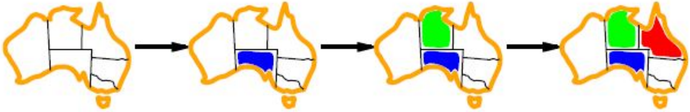


Degree heuristic

Tie-breaker among MRV variables

Degree heuristic:

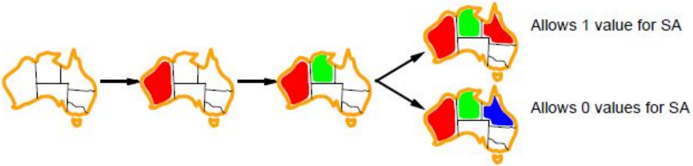
choose the variable with the most constraints on remaining variables



Backtracking Search for CSPs

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

Backtracking Search for CSPs

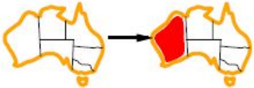
Forward checking

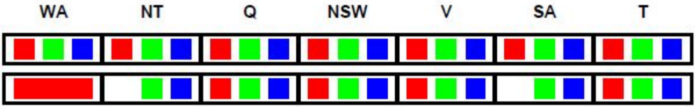
Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values

Backtracking Search for CSPs

Forward checking

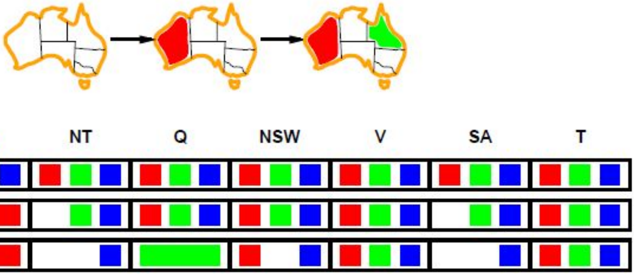
Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values





Forward checking

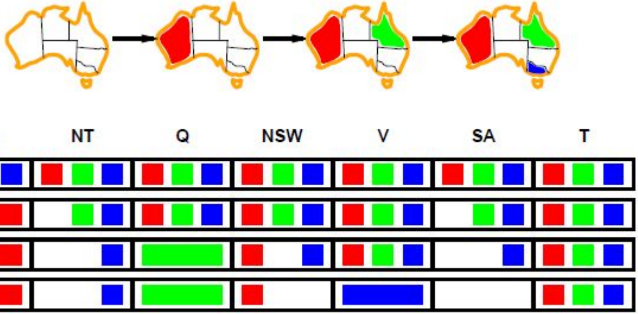
Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values



Backtracking Search for CSPs

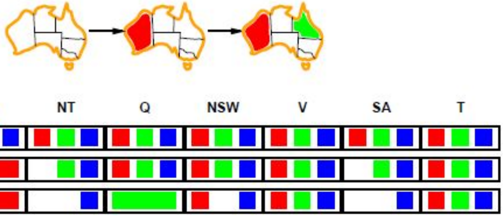
Forward checking

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:



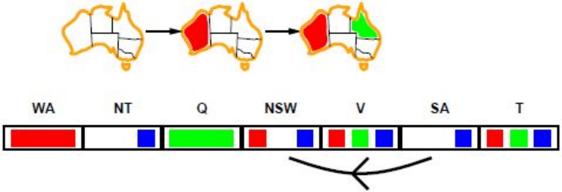
NT and SA cannot both be blue!

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



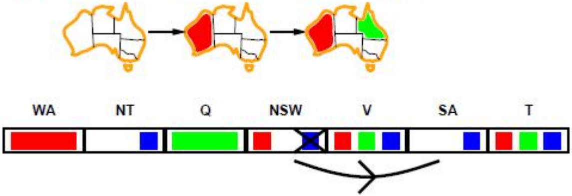
Backtracking Search for CSPs

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

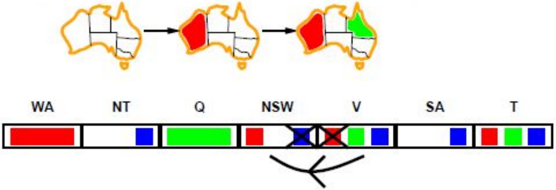


Arc consistency

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

Adversarial Search - Games

- Distinction between **cooperative** and **competitive** multiagent environments
- **Competitive environments**, in which the agents' goals are in **conflict**, give rise to **adversarial search** problems-often known as **games**.
- It means **deterministic**, **fully observable** environments in which there are **two agents** whose actions must **alternate** and in which the **utility values** at the **end of the game** are always **equal** and **opposite**.
  - If one player **wins** a game of chess (+1), the other player necessarily **loses** (-1).
  - **Chess** has an **average branching factor** of about **35** and games often go to **50 moves** by each player, so the search tree has about $35^{(50+50)}$ or 10^{154} nodes (although the search graph has "only" about 10^{40} distinct nodes).

Adversarial Search - Games

- **Pruning** allows us to **ignore portions** of the search tree that make no difference to the final choice and heuristic **evaluation function** allows us to approximate the true utility of a state without doing a complete search.
- We will consider games with **two players**, whom we will call **MAX** and **MIN**.
- **MAX** moves first and then they take turns moving until **the game is over**.
- At the **end of the game**, **points are awarded** to the **winning player** and **penalties** are given to the **loser**.

Adversarial Search - Games

- The **initial state**, which includes the board position and identifies the player to move.
- A **successor function**, which returns a list of (move, state) pairs, each indicating a **legal move** and the **resulting state**.
- **Terminal test** determines when the game is over. States where the game has ended are called **terminal states**.
- A **utility function** (also called an **objective function** or **payoff function**), which gives a **numeric value** for the terminal states. In chess, the outcome is a **win**, **loss**, or **draw**, with values **+1**, **-1**, or **0**.
- Some games have a wider, variety of possible outcomes; the **payoffs in backgammon** range from **+192** to **-192**.

Thank You!