Outline

• Constraint Satisfaction Problems (CSP)

• Backtracking search for CSPs
Outline

• Constraint Satisfaction Problems (CSP)
• Backtracking search for CSPs
• Local search for CSPs
Constraint satisfaction problems (CSPs)

- 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**: $D_i = \{\text{red, green, blue}\}$
- **Constraints**: adjacent regions must have different colors
- e.g., $WA \neq NT$, or $(WA, NT)$ in $\{(\text{red, green}), (\text{red, blue}), (\text{green, red}), (\text{green, blue}), (\text{blue, red}), (\text{blue, green})\}$
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 CSPs

• Discrete variables
  – finite domains:
    • $n$ variables, domain 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., $StartJob_1 + 5 \leq StartJob_3$

• Continuous variables
  – linear constraints solvable in polynomial time by linear programming
Varieties of constraints

- **Unary** constraints involve a single variable,
  - e.g., SA ≠ green

- **Binary** constraints involve pairs of variables,
  - e.g., SA ≠ WA

- **Higher-order** constraints involve 3 or more variables,
  - e.g., cryptarithmetic column constraints
Example: Cryptarithmetic

- **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:** \textit{Alldiff} $(F, T, U, W, R, O)$
  - $O + O = R + 10 \cdot X_1$
  - $X_1 + W + W = U + 10 \cdot X_2$
  - $X_2 + T + T = O + 10 \cdot X_3$
  - $X_3 = F, T \neq 0, F \neq 0$
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
Standard search formulation (incremental)

Let's start with the straightforward approach, then fix it

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 current assignment
  → fail if no legal assignments
- **Goal test**: the current assignment is complete

1. This is the same for all CSPs
2. Every solution appears at depth $n$ with $n$ variables
   → use depth-first search
3. Path is irrelevant, so can also use complete-state formulation
4. $b = (n - l)d$ at depth $l$, hence $n! \cdot d^n$ leaves
Backtracking search

- Variable assignments are **commutative**, i.e.,
  \[ WA = \text{red then NT} = \text{green} \] same as \[ NT = \text{green then WA} = \text{red} \]

- Only need to consider assignments to a single variable at each node
  \( b = d \) 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

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 ≠ failure 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

• 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
- $X \rightarrow Y$ 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 $y$
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.
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
- Arc consistency detects failure earlier than forward checking
- Can be run as a preprocessor or after each assignment
Arc consistency algorithm AC-3

function AC-3(csp) returns the CSP, possibly with reduced domains
input: 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
    \((X_i, X_j) \leftarrow \text{Remove-First}(queue)\)
    if RM-INCONSISTENT-VALUES\((X_i, X_j)\) then
        for each \(X_k\) in \text{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 ← false
for each \(x\) in \text{Domain}[X_i] do
    if no value \(y\) in \text{Domain}[X_j] allows \((x, y)\) to satisfy constraint\((X_i, X_j)\)
        then delete \(x\) from \text{Domain}[X_i]; removed ← true
return removed

• Time complexity: \(O(n^2d^3)\)
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