Material

• We will spend a bit more time on CSP since it is an important problem, starting with chapter 5.
• Read all of chapter 5
• Reminder: Homework: Due Wed, April 20th.
  – Do: Problems 3.4, 3.7 3.10, 4.7, and 4.16.
    • for 4.16, do both 8-puzzle & 8-queens, and also implement the two simple heuristics h1 and h2 given in the book. Compare all of them.
    • Also for 4.16, devise your own novel heuristics for A* for the 8-puzzle & 8-queens problem.
Outline For Today

• Chapter 5, Sections 1-3.
• Constraint Satisfaction Problems (CSP)
• Backtracking search for CSPs
• Local search for CSPs
Thought Question

- Let us assume that there are five houses of different colors next to each other on the same road. In each house lives a man of a different nationality. Every man has his favorite drink, his favorite brand of cigarettes, and keeps pets of a particular kind.
  - The Englishman lives in the red house.
  - The Swede keeps dogs.
  - The Dane drinks tea.
  - The green house is just to the left of the white one.
  - The owner of the green house drinks coffee.
  - The Pall Mall smoker keeps birds.
  - The owner of the yellow house smokes Dunhills.
  - The man in the center house drinks milk.
  - The Norwegian lives in the first house.
  - The Blend smoker has a neighbor who keeps cats.
  - The man who smokes Blue Masters drinks bier.
  - The man who keeps horses lives next to the Dunhill smoker.
  - The German smokes Prince.
  - The Norwegian lives next to the blue house.
  - The Blend smoker has a neighbor who drinks water.

- Question for you to answer: Who keeps the fish?
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

- This is a simple example of a formal representation language

- Allows useful general-purpose algorithms with more power than standard search algorithms (because of the special nature of the search space)
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 ≠ NT, or (WA, NT) in \{(red, green), (red, blue), (green, red), (green, blue), (blue, red), (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 $d \to O(d^n)$ 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 in general
      – e.g., $\text{StartJob}_1 + 5 \leq \text{StartJob}_3$

• Continuous variables
  – e.g., start/end times for Hubble Space Telescope observations
  – linear constraints solvable in polynomial time by linear programming
Varieties of constraints

• **Unary** constraints involve a single variable,
  – e.g., \( SA \neq \text{green} \)

• **Binary** constraints involve pairs of variables,
  – e.g., \( SA \neq WA \)

• **Higher-order** constraints involve 3 or more variables,
  – e.g., cryptarithmetic column constraints
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: $\text{Alldiff} (F, T, U, W, R, O)$
- $O + O = R + 10 \cdot X_1$
- $X_1 + W + W = U + 10 \cdot X_2$
- $X_2 + T + T = O + 10 \cdot X_3$
- $X_3 = F, T \neq 0, F \neq 0$
Real-world CSPs

- Assignment problems
  - e.g., who teaches what class
- Timetabling problems
  - e.g., which class is offered when and where?
- Hardware configuration
- Spreadsheets
- Transportation scheduling
- Factory scheduling
- Floorplanning
- Even, Bayesian Belief Propagation!!
- 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. Naive approach: \( b = (n - l)d \) at depth \( l \), hence \( n! \cdot d^n \) leaves 😞
Backtracking search

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

- Only need to consider assignments to a single variable at each node
  \[ b = d \] and there are \( d^n \) leaves

- 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 solution/failure
    return Recursive-Backtracking({}, csp)

function Recursive-Backtracking(assignment, csp) returns soln/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 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 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 better advantage of the problem structure?
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 (degree heuristic)
Least constraining value

• Given a variable, choose the least constraining value:
  – the one that rules out the fewest values in the remaining variables (useful when we want only to find one set of sat. values, not all)

• 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 $y$

We have that $SA \rightarrow NSW$
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 \( y \).

We do not have that NSW \( \rightarrow \) SA.
Arc consistency

- Simplest form of propagation makes each arc consistent
- $X \rightarrow Y$ is consistent iff
  
  \[
  \text{for every value } x \text{ of } X \text{ there is some allowed } y
  \]

- If $X$ loses a value, neighbors of $X$ need to be rechecked

Removing blue from NSW means that we no longer have that $V \rightarrow NSW$
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
inputs: 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) ← REMOVE-FIRST(queue)
    if REMOVE-INCONSISTENT-VALUES(X_i, X_j) then
        for each X_k in NEIGHBORS[X_i] do
            add (X_k, X_i) to queue

function REMOVE-INCONSISTENT-VALUES(X_i, X_j) returns true iff succeeds
removed ← false
for each x in DOMAIN[X_i] do
    if no value y in DOMAIN[X_j] allows (x,y) to satisfy the constraint X_i \leftrightarrow X_j
    then delete x from DOMAIN[X_i]; removed ← true
return removed

• Time complexity: O(n^2d^3)  (AC-4 is O(n^2d^2))
Local search for CSPs

• Hill-climbing, simulated annealing typically work with "complete" states, i.e., all variables assigned
• To apply to CSPs:
  – allow states with unsatisfied constraints
  – operators reassign variable values (perhaps even randomly change state)
• Variable selection: randomly select any conflicted variable and change its value.
• Value selection by \textbf{min-conflicts} heuristic:
  – choose value that violates the fewest constraints
  – i.e., hill-climb with $h(n) = \text{total number of violated constraints}$
Example: 4-Queens

- **States**: 4 queens in 4 columns \((4^4 = 256\) states)
- **Actions**: move queen in column
- **Goal test**: no attacks
- **Evaluation**: \(h(n) = \text{number of attacks}\)

Given random initial state, can solve \(n\)-queens in almost constant time for arbitrary \(n\) with high probability (e.g., \(n = 10,000,000\))
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

• Iterative min-conflicts is usually effective in practice