Material

- Read all of chapter 5
- Reminder: Homework: Due Today.
  - 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

- Rest of Chapter 5

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, with values from domain D,
  - 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 = {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

Backtracking search

```
function BACKTRACKING-SEARCH() returns solution/failure
return RECURSIVE-BACKTRACKING()

function RECURSIVE-BACKTRACKING(assignment, nogood) returns solution/failure
if assignment is complete then return assignment
var ← SELECT-UNASSIGNED-VARIABLE(VALUES[assignment])
for each value in VALUES[var] do
    if value is consistent with assignment then
        add (var, value) to assignment
        result ← RECURSIVE-BACKTRACKING(assignment, nogood)
        if result is solution then return result
    remove (var, value) from assignment
return failure
```

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

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
Arc consistency

- Simplest form of propagation makes each arc consistent
- X \rightarrow Y is consistent if 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

\begin{align*}
\text{function AC-3}(\varphi) & \text{ returns the CSP, possibly with reduced domains} \\
\text{inputs: } & \varphi, \text{ a binary CSP with variables } \{X_1, X_2, \ldots, X_r\} \\
\text{local variables: } & \text{queue, a queue of arcs, initially all the arcs in } \varphi \\
\text{while } & \text{queue is not empty } \\
\text{do } & (X_i, X_j) \leftarrow \text{REMOVE-FIRST}(\text{queue}) \\
\text{if } & \text{Remove-Inconsistent-Value}(X_i, X_j) \text{ then} \\
\text{for each } & X_k \text{ in } \text{Neighbors}(X_j) \text{ do} \\
& \text{add } (X_k, X_j) \text{ to queue} \\
\text{function Remove-Inconsistent-Value}(X_i, X_j) & \text{ returns true if succeeds} \\
& \text{removed} \leftarrow \text{false} \\
& \text{for each } x \text{ in } \text{Domain}[X_i] \text{ do} \\
& \text{if no } y \text{ in } \text{Domain}[X_j] \text{ allows } (x, y) \text{ to satisfy the constraint } X_i \rightarrow X_j \\
& \text{then define } x \text{ from } \text{Domain}[X_i] \text{ removed} \leftarrow \text{true} \\
& \text{return } \text{removed} \\
\end{align*}

- Time complexity: O(n^2d^3) (AC-4 is O(n^2d^2))

Stronger Forms of Consistency

- Def: k-consistency: A CSP is k-consistent if for any set of k-1 variables and for any consistent assignment to those variables, a consistent value can always be assigned to any kth variable. Example:
  - 1-consistency (node-consistency): each variable is consistent itself
  - 2-consistency: same as arc-consistency
  - 3-consistency: (path-consistency), any pair of adjacent variables can always be extended to third variable
- Strongly k-consistent: if it is j-consistent, for j = k down to 1.
  - Note: if CSP is strongly n-consistent, then we can solve problem in O(nd) time (choose a value for first variable, then choose a value for some second one, etc.)

Backtracking search

\begin{align*}
\text{function Backtracking-Search}(\varphi) & \text{ returns solution, failure} \\
\text{return } & \text{RECURSIVE-BACKTRACKING}(\{\}, \varphi) \\
\text{function RECURSIVE-BACKTRACKING}(assign, \varphi) & \text{ returns solution, failure} \\
\text{if } & \text{assignment is complete they return assignment} \\
& \text{var} \leftarrow \text{SELECT-UNASSIGNED-VARIABLE}(\text{variables}[\varphi], \text{assignment}, \varphi) \\
& \text{for each value } v \text{ in } \text{Domain}[\text{var}] \text{ do} \\
& \text{if value is consistent with assignment gives } \text{Constraints}()[\varphi] \text{ then} \\
& \text{add } \{\text{var} \leftarrow \text{value}\} \text{ to assignment} \\
& \text{result } \leftarrow \text{RECURSIVE-BACKTRACKING}(\text{assignment}, \varphi) \\
& \text{if result is failure then return result} \\
& \text{return } \{\text{var} \leftarrow \text{value}\} \text{ from assignment} \\
& \text{return } \text{failure} \\
\end{align*}

Intelligent Backjumping

- Suppose we search using order: Q, NSW, V, T, SA and have assignment: red, green, blue, red respectively.
- We go to assign SA, nothing works, so back up to T and try blue, green for T.
- But this is wasteful since changing T is not going to change the values of the variables constraining SA.
- Intelligent backjumping is how we get around this.
Intelligent Backjumping

- Forward checking also can compute conflict set
- In fact, forward checking and backjumping can be redundant

Conflict-directed backjumping

- Reason for failure might not be detected.
- WA.NSW 1st to assign and are red, SA is last. In this order, all colors of NT will fail, but NT is consistent (nothing wrong).
- We keep track of children’s conflict set and propagate it up so we know what cause of problem is: Variable $X_i$ and most recent conflict is $X_{\pi(i)}$
- Do:
  \[
  \text{conf}(X_i) \leftarrow \text{conf}(X_i) \cup \text{conf}(X_{\pi(i)}) - \{X_i\}
  \]

Local (iterative) 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 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., $n = 10,000,000$), except ...

Performance of min-conflicts

- Given random initial state, can solve $n$-queens in almost constant time for arbitrary $n$ with high prob.
- This appears to be true for any randomly-generated CSP except in a narrow range of the ratio:
  \[
  R = \frac{\text{number of constraints}}{\text{number of variables}}
  \]

Problem Structure

- Tasmania and mainland are independent subproblems
- Identifiable as connected components of constraint graph
Problem Structure

• Independent problems can be solved, well, independently.
• Example: if each sub-problem has c out of n variables, worse case solution will be \( O((n/c)^d^c) \) which is linear in n
  – this can be a big constant, but in general it is much better to solve independent problems separately.
  – Example: \( n=80, \ d=2, \ c=20 \)
    • \( 2^80 \approx 4 \text{ billion years at 10 million nodes/sec} \)
    • \( 4\cdot 2^{20} \approx 0.4 \text{ seconds at 10 million nodes/sec} \)
    • only 0.1 seconds of we exploit inherent parallelism in problem.

Tree-structured CSPs

• Thm: if the constraint graph has no loops (i.e., a tree), CSP can be solved in \( O(n^d^2) \) time.
• Compare this to general worse-case time \( O(d^n) \)
• Consider algorithm:
  1) choose any var as root, label as \( X_1 \). Label all other variables so that vars closer to root have lower numbers than those farther away. This gives us a directed acyclic graph (DAG) with variables having at most one parent each.

Tree-structured CSPs

– 2) from leaves moving up to root, apply arc consistency between parent and child. I.e., call RemoveInconsistentParent(\( X_\pi(j), X_j \))
– 3) Starting from root, choose a consistent assignment, then for each child choose a consistent assignment (guaranteed to exist), and keep doing down tree until you hit leaves.
• Due to the directional arc-consistency, this is guaranteed to work!

Nearly Tree-structured CSPs

• Conditioning: instantiate a variable, prune its neighbors domains:
• Custset conditioning: instantiate (in all ways) a set of variables such that the remaining constant graph is a tree.
• Custset size \( c \) runtime \( O(df(n-c)d^2) \), fast if \( c \) is small!!

Nearly Tree-structured CSPs

• General Algorithm:
  – Choose a subset \( S \) from VARIABLES[csp] such that graph becomes a tree after removal of \( S \). \( S \) is called a cycle cutset. It has \( c \) variables
  – For each possible assignment of vars in \( S \) (\( O(d^c) \)) satisfying constraints on \( S \), do:
    • remove from remaining variables values that are inconsistent with assignments for \( S \), and
    • If remaining CSP has a solution (easy to find since remaining vars is a tree), return it together with the assignment to \( S \)
• Problems:
  – custset might be as large as \( n-2 \) variables.
  – when? Consider a clique
  – Finding smallest cycle cutset is NP-hard. Overall approach is called cutset conditioning, originally developed for probabilistic inference.

Nearly Tree-structured CSPs

• We can characterize how difficult a given CSP problem might be using a generalized notion of "tree", called a "k-tree"
• A 1-tree is just a normal tree:
  – a graph that has no cycles
  – a graph where there is one and only one path that connects any two arbitrarily chosen nodes.
  – A path is also a tree.
Nearly Tree-structured CSPs

• A k-tree is defined as follows:
  – Any fully connected set of (k+1) nodes is a k-tree
  – Given any k-tree on n nodes, we can get another k-tree with (n+1) nodes, by connecting
    that new node to a set k other nodes that are fully connected.

• Example: 2-trees

• Thm: We can solve any CSP on a k-tree in time O(n^{d^{k+1}})

Summary Of Today

• 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
• For Next time: Read Bacchus Paper!!