CS 188: Artificial Intelligence
Constraint Satisfaction Problems III
Instructor: Marco Alvarez
University of Rhode Island

Today

- Efficient Solution of CSPs
- Local Search

K-Consistency

- Strong K-Consistency
  - Strong k-consistency: also k-1, k-2, ... 1 consistent
  - Claim: strong n-consistency means we can solve without backtracking!
  - Why?
    - Choose any assignment to any variable
    - Choose a new variable
    - By 2-consistency, there is a choice consistent with the first
    - Choose a new variable
    - By 3-consistency, there is a choice consistent with the first 2
    - ...
  - Lots of middle ground between arc consistency and n-consistency! (e.g. k=3, called path consistency)

Improving Backtracking

- Filtering: Can we detect inevitable failure early?
- Ordering:
  - Which variable should be assigned next? (MRV)
  - In what order should its values be tried? (LCV)
- Structure: Can we exploit the problem structure?

Ordering: Minimum Remaining Values

- Variable Ordering: Minimum remaining values (MRV):
  - Choose the variable with the fewest legal left values in its domain
  - Why min rather than max?
  - Also called “most constrained variable”
  - “Fail-fast” ordering

Ordering: Least Constraining Value

- Value Ordering: Least Constraining Value
  - Given a choice of variable, choose the least constraining value
    - i.e., the one that rules out the fewest values in the remaining variables
  - Note that it may take some computation to determine this! (E.g., rerunning filtering)
  - Why least rather than most?
  - Combining these ordering ideas makes 1000 queens feasible
Improving Backtracking

- Filtering: Can we detect inevitable failure early?
- Ordering:
  - Which variable should be assigned next? (MRV)
  - In what order should its values be tried? (LCV)
- Structure: Can we exploit the problem structure?

Structure

- Extreme case: independent subproblems
  - Example: Tasmania and mainland do not interact
- Independent subproblems are identifiable as connected components of constraint graph
- Suppose a graph of n variables can be broken into subproblems of only c variables:
  - Worst-case solution cost is \(O(n/c \cdot d^c)\), linear in n
  - E.g., \(n = 80, d = 2, c = 20\)
  - \(2^{80}/2^{20} = 4\) billion years at 10 million nodes/sec
    - \(4 \cdot 2^{20} = 0.4\) seconds at 10 million nodes/sec

Problem Structure

- Tree-Structured CSPs
  - Algorithm for tree-structured CSPs:
    - Order: Choose a root variable, order variables so that parents precede children
    - Remove backward: For \(i = n : 2\), apply RemoveInconsistent(Parent(X_i), X_i)
    - Assign forward: For \(i = 1 : n\), assign \(X_i\) consistently with Parent(X_i)
  - Runtime: \(O(n \cdot d^2)\) (why?)

- Claim 1: After backward pass, all root-to-leaf arcs are consistent
  - Proof: Each \(X \rightarrow Y\) was made consistent at one point and Y’s domain could not have been reduced thereafter (because Y’s children were processed before Y)

- Claim 2: If root-to-leaf arcs are consistent, forward assignment will not backtrack
  - Proof: Induction on position

- Why doesn’t this algorithm work with cycles in the constraint graph?
- Note: we’ll see this basic idea again with Bayes’ nets

Improving Structure

- Nearly Tree-Structured CSPs
  - Conditioning: instantiate a variable, prune its neighbors’ domains
  - Cutset conditioning: instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree
  - Cutset size \(c\) gives runtime \(O\left(d^c \cdot (n-c) \cdot d^c\right)\), very fast for small \(c\)

Cutset Conditioning

- Choose a cutset
- Instantiate the cutset (all possible ways)
- Compute residual CSP for each assignment
- Solve the residual CSPs (tree-structured)
Cutset Quiz

- Find the smallest cutset for the graph below.

Iterative Improvement

- Local search methods typically work with "complete" states, i.e., all variables assigned
- To apply to CSPs:
  - Take an assignment with unsatisfied constraints
  - Operators reassign variable values
  - No fringe! Live on the edge.
- Algorithm: While not solved,
  - Variable selection: randomly select any conflicted variable
  - Value selection: min-conflicts heuristic:
    - Choose a 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)
- Operators: move queen in column
- Goal test: no attacks
- Evaluation: \( c(n) = \text{number of attacks} \)

Video of Demo Iterative Improvement - n Queens

Performance of Min-Conflicts

- Given random initial state, can solve n-queens in almost constant time for arbitrary \( n \) with high probability (e.g., \( n = 10,000,000 \)!
- The same appears to be true for any randomly-generated CSP except in a narrow range of the ratio \( R = \frac{\text{number of constraints}}{\text{number of variables}} \)

Summary: CSPs

- CSPs are a special kind of search problem:
  - States are partial assignments
  - Goal test defined by constraints
- Basic solution: backtracking search
- Speed-ups:
  - Ordering
  - Filtering
  - Structure
- Iterative min-conflicts is often effective in practice

Local Search
Local Search

- Tree search keeps unexplored alternatives on the fringe (ensures completeness)
- Local search: improve a single option until you can’t make it better (no fringe!)
- New successor function: local changes
- Generally much faster and more memory efficient (but incomplete and suboptimal)

Hill Climbing

- Simple, general idea:
  - Start wherever
  - Repeat: move to the best neighboring state
  - If no neighbors better than current, quit
- What’s bad about this approach?
  - Complete?
  - Optimal?
- What’s good about it?

Genetic Algorithms

- Genetic algorithms use a natural selection metaphor
- Keep best N hypotheses at each step (selection) based on a fitness function
- Also have pairwise crossover operators, with optional mutation to give variety
- Possibly the most misunderstood, misapplied (and even maligned) technique around

Hill Climbing Quiz

Starting from X, where do you end up?
Starting from Y, where do you end up?
Starting from Z, where do you end up?

Simulated Annealing

- Idea: Escape local maxima by allowing downhill moves but make them rarer as time goes on

Example: N-Queens

- Why does crossover make sense here?
- When wouldn’t it make sense?
- What would mutation be?
- What would a good fitness function be?

Simulated Annealing

- Theoretical guarantee:
  - Stationary distribution: \( p(x) \propto e^{-\frac{E(x)}{T}} \)
  - If T decreased slowly enough, will converge to optimal state!
- Is this an interesting guarantee?
- Sounds like magic, but reality is reality:
  - The more downhill steps you need to escape a local optimum, the less likely you are to ever make them all in a row
  - People think hard about ridge operators which let you jump around the space in better ways