Announcements

- Slides, notes, books
  - Slides are always available before class
  - You may want to print them out and take notes on them!

- Midterm: 10/10
  - Cheat sheet only, one page, make your own

- Need a partner? Stay after class.

- Assignment 1.2 is up (due 9/19)

Today

- More CSPs
  - Applications
  - Tree Algorithms
  - Cutset Conditioning

- Local Search

Reminder: CSPs

- CSPs:
  - Variables
  - Domains
  - Constraints
    - Implicit (provide code to compute)
    - Explicit (provide a subset of the possible tuples)

- Unary Constraints
- Binary Constraints
- N-ary Constraints

Example: Boolean Satisfiability

- Given a Boolean expression, is it satisfiable?
- Very basic problem in computer science

\[ p_1 \land (p_2 \rightarrow p_3) \land ((\neg p_1 \land \neg p_3) \rightarrow \neg p_2) \land (p_1 \lor p_3) \]

- Turns out you can always express in DNF

\[ (p_1) \land (\neg p_2 \lor p_3) \land (p_1 \lor p_3 \lor \neg p_2) \land (p_1 \lor p_2 \lor p_3) \]

- D SAT: find a satisfying truth assignment

Example: 3-SAT

- Variables: \( p_1, p_2, \ldots, p_n \)
- Domains: \{true, false\}
- Constraints: \( p_i \lor p_j \lor p_k \)

\[ \neg p_q \lor p_j \lor p_k \]

Implicitly conjoined (all clauses must be satisfied)
CSPs: Queries

- Types of queries:
  - Legal assignment (last class)
  - All assignments
  - Possible values of some query variable(s) given some evidence (partial assignments)

Example: Fault Diagnosis

- Fault networks:
  - Variables?
  - Domains?
  - Constraints?
- Various ways to query, given symptoms
  - Some cause (abduction)
  - Simplest cause
  - All possible causes
  - What test is most useful?
  - Prediction: cause to effect

- We’ll see this idea again with Bayes’ nets

Example: N-Queens

- Formulation 2:
  - Variables: $Q_k$
  - Domains: $\{11, 12, 13, \ldots, 21, \ldots, NV\}$
  - Constraints:
    \[
    \forall i, j \ (Q_i, Q_j) \in \{(11, 23), (11, 24), \ldots\} \\
    \forall i, j \text{ non-threatening}(Q_i, Q_j) \\
    \]

Reminder: Consistency

- Basic solution: DFS / backtracking
  - Add a new assignment
  - Check for violations
  - Forward checking:
    - Pre-filter unassigned domains after every assignment
    - Only remove values which conflict with current assignments
  - Arc consistency
    - We only defined it for binary CSPs
    - Check for impossible values on all pairs of variables, prune them
    - Run (or not) after each assignment before recursing
    - A pre-filter, not search!

Limitations of Arc Consistency

- After running arc consistency:
  - Can have one solution left
  - Can have multiple solutions left
  - Can have no solutions left (and not know it)

What went wrong here?
**K-Consistency**

- Increasing degrees of consistency
  - 1-Consistency (Node Consistency): Each single node’s domain has a value which meets that node’s unary constraints
  - 2-Consistency (Arc Consistency): For each pair of nodes, any consistent assignment to one can be extended to the other
  - K-Consistency: For each k nodes, any consistent assignment to k-1 can be extended to the kth node.
- Higher k more expensive to compute
- (You need to know the k=2 algorithm)

**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. path consistency)

**Problem Structure**

- Tasmania and mainland are independent subproblems
- Identifiable as connected components of constraint graph
- Suppose each subproblem has c variables out of n total
  - Worst-case solution cost is \(O(n(c^2))\), linear in n
  - E.g., n = 80, d = 2, c = 20
  - \(2^{40} = 4 \text{ billion years at 10 million nodes/sec}\)
  - \((4)(2^{20}) = 0.4 \text{ seconds at 10 million nodes/sec}\)

**Tree-Structured CSPs**

- Choose a variable as root, order variables from root to leaves such that every node’s parent precedes it in the ordering
- For \(i = n : 2\), apply \(\text{RemoveInconsistent(Parent}(X_i),X_i)\)
- For \(i = 1 : n\), assign \(X_i\) consistently with \(\text{Parent}(X_i)\)
- Runtime: \(O(n^2)\) (why?)

Why does this work?

- Claim: After each node is processed leftward, all nodes to the right can be assigned in any way consistent with their parent.
- Proof: Induction on position

Why doesn’t this algorithm work with loops?

- Note: we’ll see this basic idea again with Bayes’ nets and call it belief propagation
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 $O((d^c)(n-c)d^2)$, very fast for small $c$

Types of Problems

- Planning problems:
  - We want a path to a solution (examples?)
  - Usually want an optimal path
  - Incremental formulations
- Identification problems:
  - We actually just want to know what the goal is (examples?)
  - Usually want an optimal goal
  - Complete-state formulations
  - Iterative improvement algorithms

Iterative Algorithms 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
- Variable selection: randomly select any conflicted variable
- Value selection by min-conflicts heuristic:
  - Choose value that violates the fewest constraints
  - I.e., hillclimb with $h(n) =$ 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: $h(n) =$ number of attacks

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 $R \approx \frac{\text{number of constraints}}{\text{number of variables}}$

CSP Summary

- CSPs are a special kind of search 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 legal 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
  - The constraint graph representation allows analysis of problem structure
  - Tree-structured CSPs can be solved in linear time
  - Iterative min-conflicts is usually effective in practice
Local Search Methods

- Queue-based algorithms keep fallback options (backtracking)
- Local search: improve what you have until you can’t make it better
- Generally much more efficient (but incomplete)

Hill Climbing

- Simple, general idea:
  - Start wherever
  - Always choose the best neighbor
  - If no neighbors have better scores than current, quit
- Why can this be a terrible idea?
  - Complete?
  - Optimal?
  - What’s good about it?

Hill Climbing Diagram

- Random restarts?
- Random sideways steps?

Simulated Annealing

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

```python
def simulated_annealing(problem, schedule):
    current = make_node(initial_state(problem))
    T = schedule(1)
    for t := 1 to ∞ do
        if T = 0 then return current
        move = a randomly selected successor of current
        ΔE = value[move] - value[current]
        if ΔE > 0 then current := move
        else current := current with probability e^(-ΔE/T)
    T := schedule(T)
```

- Theoretical guarantee:
  - Stationary distribution: \( p(x) \propto e^{-E(x)/T} \)
  - 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, the less likely you are to every make them all in a row
  - People think hard about ridge operators which let you jump around the space in better ways

Beam Search

- Like greedy search, but keep K states at all times:
  - Variables: beam size, encourage diversity?
  - The best choice in MANY practical settings
  - Complete? Optimal?
  - Why do we still need optimal methods?
**Genetic Algorithms**

- Genetic algorithms use a natural selection metaphor
- Like beam search (selection), but also have pairwise crossover operators, with optional mutation
- Probably the most misunderstood, misapplied (and even maligned) technique around!

**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?

**Continuous Problems**

- Placing airports in Romania
  - States: \((x_1, y_1, x_2, y_2, x_3, y_3)\)
  - Cost: sum of squared distances to closest city

**Gradient Methods**

- How to deal with continuous (therefore infinite) state spaces?
- Discretization: bucket ranges of values
  - E.g. force integral coordinates
- Continuous optimization
  - E.g. gradient ascent
  \[
  \nabla f = \left( \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3} \right)
  \]
  \[
  x \leftarrow x + \alpha \nabla f(x)
  \]