CSPs:
- Variables
- Domains
- Constraints
- Implicit (provide code to compute)
- Explicit (provide a list of the legal tuples)
- Unary / Binary / N-ary
Goals:
- Here: find any solution
- Also: find all, find best, etc.

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 \( O((n/c)(d^c)) \), linear in n
- E.g., n = 80, d = 2, c = 20
- \( 280 \times 4 \) billion years at 10 million nodes/sec
- \( (4)(2^{20}) = 0.4 \) seconds at 10 million nodes/sec

Algorithm for tree-structured CSPs:
- Order: Choose root variable and order variables so that parent precedes children
  - For \( i = n \); 2, apply RemoveInconsistent(Parent(X), X)
  - Assign forward:
    - For \( i = 1 : n \), assign \( X_i \) consistently with Parent(\( X_i \))
- Runtime: \( O(nd^2) \) (why?)
**Tree-Structured CSPs**

Claim 1: After backward pass, all root-to-leaf arcs are consistent

Proof: Each X ← 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 $O((d^c)(n − c)d^2)$, 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.

**Tree Decomposition**

Idea: create a tree-structured graph of mega-variables

Each mega-variable encodes part of the original CSP

Subproblems overlap to ensure consistent solutions

Agree:

$(M_1, M_2) \in \{((WA = r, NT = g, SA = b), \ldots), ((NT = r, SA = g, Q = b), \ldots)\}$
Iterative Algorithms for CSPs

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 \( h(n) = \) 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: \( c(n) = \) number of attacks

Demo: n-queens – iterative improvement (L5D1) Demo: coloring – iterative improvement

Performance of Min-Conflicts

Given random initial state, can solve n-queens in almost linear 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**

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?

---

**Hill Climbing Diagram**

---

**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**

Theoretical guarantee:
- Stationary distribution: \( p(x) \propto e^{E(x)/kT} \)
- If \( T \) decreased slowly enough, will converge to optimal state!

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

**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

**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?

**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.

**Beam Search**

Like greedy hillclimbing 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?
Greedy Search
Beam Search
CSP Formulation: 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.

Next Time: Adversarial Search!

Best strategy against opponent.