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: The Waltz Algorithm

- The Waltz algorithm is for interpreting line drawings of solid polyhedra
- An early example of a computation posed as a CSP

Look at all intersections
Adjacent intersections impose constraints on each other

Waltz on Simple Scenes

- Assume all objects:
  - Have no shadows or cracks
  - Three-faced vertices
  - "General position": no junctions change with small movements of the eye.
- Then each line on image is one of the following:
  - Boundary line (edge of an object) (→) with right hand of arrow denoting "solid" and left hand denoting "space"
  - Interior convex edge (+)
  - Interior concave edge (-)

Legal Junctions

- Only certain junctions are physically possible
- How can we formulate a CSP to label an image?
- Variables: vertices
- Domains: junction labels
- Constraints: both ends of a line should have the same label
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 3-CNF
  \[ (p_1) \land (\neg p_2 \lor p_3) \land (p_1 \lor p_3 \lor \neg p_2) \land (p_1 \lor p_2 \lor p_3) \]
- 3-SAT: find a satisfying truth assignment

Example: 3-SAT
- Variables: \( p_1, p_2, \ldots, p_n \)
- Domains: \{ true, false \}
- Constraints: \[ p_1 \lor p_2 \lor p_3 \]
  \[ \lor \cdots \lor p_k \]
  \[ \lor \neg p_1 \lor \neg p_2 \lor \neg p_3 \]
  \[ \lor \cdots \lor \neg p_k \]

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

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
  - Run (or not) after each assignment before recursing

Arc Consistency

```
function ACSR(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-VALUE \( (X_i, X_j) \) then
    for each \( x_j \) in DOMAIN \( X_j \) do
      add \( (X_i, x_j) \) to queue

function REMOVE-INCONSISTENT-VALUE \( (X_i, X_j) \) returns true if succeeds
removed = false
for each \( x_i \) in DOMAIN \( X_i \) do
  if no value \( x_j \) in DOMAIN \( X_j \) allows \( x_i \) to satisfy the constraint \( X_i \rightarrow X_j \) then
    remove \( x_i \) from DOMAIN \( X_i \)
    removed = true
return removed
```
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

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

Tree-Structured CSPs

- Theorem: if the constraint graph has no loops, the CSP can be solved in $O(n d^2)$ time
  - Compare to general CSPs, where worst-case time is $O(d^n)$

- This property also applies to logical and probabilistic reasoning: an important example of the relation between syntactic restrictions and the complexity of reasoning.

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 RemoveInconsistent(Parent(Xi),Xi)
- For $i = 1 : n$, assign Xi, consistently with Parent(Xi)
- Runtime: $O(n d^2)$ (why?)
Tree-Structured CSPs

- 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

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

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)

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

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 on

  \[
  p(x) \propto e^{\frac{-E(x)}{T}}
  \]

- Theoretical guarantee:
  - Stationary distribution:
  - 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:

  ![Beam Search Diagram]

- 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
  
  \[
  \nabla f = \left( \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial x_4} \right)
  \]

  \[
  x \leftarrow x + \alpha \nabla f(x)
  \]

Potential Fields

- Cost for:
  - Being far from goal
  - Being near an obstacle
- Go downhill
- What could go wrong?

Image from vias.org
Potential Field Methods

Define a function $u(q)$

$u : \text{Configurations} \rightarrow \mathbb{R}$

Such that

$u \rightarrow \text{huge}$ as you move towards an obstacle

$u \rightarrow \text{small}$ as you move towards the goal

Write $d(q) = d(q, q_{\text{goal}})$

$d'(q) = \text{distance from } q \text{ to nearest obstacle}$

One definition of $u$: $u(q) = d(q) - d_{\text{goal}}(q)$

Preferred definition: $u(q) = \frac{1}{2} \sum (d(q)^2) + \frac{1}{2} \frac{1}{d(q)^2}$

SIMPLE MOTION PLANNER:

Gradient descent on $u$

Next Time

- Probabilities (chapter 13)
- Basis of most of the rest of the course
- You might want to read up in advance!