Announcements

- **Reminder:**
  - Project 1.1 is due Friday at 11:59pm!
  - Check web page for this week’s office hours

- **Sections this Monday**
  - Can go to any of them, or multiple (unless over capacity of room)
  - Dan / John back late today

- **Don’t forget about the newsgroup**
  - Good for course questions
  - Good for finding partners
Constraint Satisfaction Problems

- **Standard search problems:**
  - State is a “black box”: any old data structure
  - Goal test: any function over states
  - Successors: any map from states to sets of states

- **Constraint satisfaction problems (CSPs):**
  - State is defined by variables $X_i$, with values from a domain $D$ (sometimes $D$ depends on $i$)
  - Goal test is a set of constraints specifying allowable combinations of values for subsets of variables

- Simple example of a formal representation language

- Allows useful general-purpose algorithms with more power than standard search algorithms

---

Example: N-Queens

- **Formulation 1:**
  - Variables: $X_{ij}$
  - Domains: $\{0, 1\}$
  - Constraints

\[
\forall i, j, k \ (X_{ij}, X_{ik}) \in \{(0, 0), (0, 1), (1, 0)\} \\
\forall i, j, k \ (X_{ij}, X_{kj}) \in \{(0, 0), (0, 1), (1, 0)\} \\
\forall i, j, k \ (X_{ij}, X_{i+k,j+k}) \in \{(0, 0), (0, 1), (1, 0)\} \\
\forall i, j, k \ (X_{ij}, X_{i+k,j}) \in \{(0, 0), (0, 1), (1, 0)\} \\
\sum_{i,j} X_{ij} = N
\]
Example: N-Queens

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

...there's an even better way! What is it?

Example: Map-Coloring

- **Variables:** $WA, NT, Q, NSW, V, SA, T$
- **Domain:** $D = \{red, green, blue\}$
- **Constraints:** adjacent regions must have different colors
  $$WA \neq NT$$
  $$(WA, NT) \in \{(red, green), (red, blue), (green, red), \ldots\}$$
- **Solutions are assignments satisfying all constraints, e.g.:**
  $$\{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green\}$$
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

Constraint Graphs

- Binary CSP: each constraint relates (at most) two variables
- Constraint graph: nodes are variables, arcs show constraints
- General-purpose CSP algorithms use the graph structure to speed up search. E.g., Tasmania is an independent subproblem!
Example: Cryptarithmetic

- **Variables:**
  \[ F \ T \ U \ W \ R \ O \ X_1 \ X_2 \ X_3 \]
  \[ + \ T \ W \ O \]
  \[ F \ O \ U \ R \]

- **Domains:**
  \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}

- **Constraints:**
  \text{alldiff}(F, T, U, W, R, O)
  \[ O \mid O = R \mid 10 \cdot X_1 \]
  \[ \ldots \]

Varieties of CSPs

- **Discrete Variables**
  - Finite domains
    - Size \( d \) means \( O(d^n) \) complete assignments
    - E.g., Boolean CSPs, including Boolean satisfiability (NP-complete)
  - Infinite domains (integers, strings, etc.)
    - E.g., job scheduling, variables are start/end times for each job
    - Need a constraint language, e.g., \( \text{StartJob}_1 + 5 < \text{StartJob}_3 \)
    - Linear constraints solvable, nonlinear undecidable

- **Continuous variables**
  - E.g., start/end times for Hubble Telescope observations
  - Linear constraints solvable in polynomial time by LP methods
    (see cs170 for a bit of this theory)
Varieties of Constraints

- **Varieties of Constraints**
  - Unary constraints involve a single variable (equiv. to shrinking domains):
    \[ SA \neq green \]
  - Binary constraints involve pairs of variables:
    \[ SA \neq WA \]
  - Higher-order constraints involve 3 or more variables:
    e.g., cryptarithmetic column constraints

- **Preferences (soft constraints):**
  - E.g., red is better than green
  - Often representable by a cost for each variable assignment
  - Gives constrained optimization problems
  - (We’ll ignore these until we get to Bayes’ nets)

Real-World CSPs

- **Assignment problems**: e.g., who teaches what class
- **Timetabling problems**: e.g., which class is offered when and where?
- **Hardware configuration**
- **Spreadsheets**
- **Transportation scheduling**
- **Factory scheduling**
- **Floorplanning**

- **Many real-world problems involve real-valued variables...**
Standard Search Formulation

- Standard search formulation of CSPs (incremental)
- Let's start with the straightforward, dumb approach, then fix it
- States are defined by the values assigned so far
  - Initial state: the empty assignment, {}  
  - Successor function: assign a value to an unassigned variable
  - Goal test: the current assignment is complete and satisfies all constraints

Search Methods

- What does BFS do?
- What does DFS do?
  - [ANIMATION]
- What’s the obvious problem here?
- What’s the slightly-less-obvious problem?
Backtracking Search

- Idea 1: Only consider a single variable at each point:
  - Variable assignments are commutative
  - I.e., [WA = red then NT = green] same as [NT = green then WA = red]
  - Only need to consider assignments to a single variable at each step
  - How many leaves are there?

- Idea 2: Only allow legal assignments at each point
  - I.e. consider only values which do not conflict previous assignments
  - Might have to do some computation to figure out whether a value is ok

- Depth-first search for CSPs with these two improvements is called backtracking search
  - [ANIMATION]

- Backtracking search is the basic uninformed algorithm for CSPs
- Can solve n-queens for $n \approx 25$

```python
function Backtracking-Search(csp) returns solution/failure
    return Recursive-Backtracking([], csp)

function Recursive-Backtracking(assignment, csp) returns solution/failure
    if assignment is complete then return assignment
    var ← Select-Unassigned-Variable(VARIABLES[csp], assignment, csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment given CONSTRAINTS[csp] then
            add {var = value} to assignment
            result ← Recursive-Backtracking(assignment, csp)
            if result ≠ failure then return result
            remove {var = value} from assignment
    return failure
```

- What are the choice points?
General-purpose ideas 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 advantage of problem structure?
Minimum Remaining Values

- Minimum remaining values (MRV):
  - Choose the variable with the fewest legal values

- Why min rather than max?
- Called most constrained variable
- “Fail-fast” ordering

Degree Heuristic

- Tie-breaker among MRV variables
- Degree heuristic:
  - Choose the variable with the most constraints on remaining variables

- Why most rather than fewest constraints?
Least Constraining Value

- Given a choice of variable:
  - Choose the least constraining value
  - The one that rules out the fewest values in the remaining variables
  - Note that it may take some computation to determine this!

- Why least rather than most?

- Combining these heuristics makes 1000 queens feasible

Forward Checking

- Idea: Keep track of remaining legal values for unassigned variables
- Idea: Terminate when any variable has no legal values
Constraint Propagation

- Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:

- NT and SA cannot both be blue!
- Why didn't we detect this yet?
- Constraint propagation repeatedly enforces constraints (locally)

Arc Consistency

- Simplest form of propagation makes each arc consistent
  - X → Y is consistent iff for every value 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
- What's the downside of arc consistency?
- Can be run as a preprocessor or after each assignment
**Arc Consistency**

```python
function AC-3(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-VALUES\(\{X_i, X_j\}\) then
        for each \(X_k\) in NEIGHBORS\(\{X_j\}\) do
            add \((X_k, X_j)\) to queue

function REMOVE-INCONSISTENT-VALUES\(\{X_i, X_j\}\) returns true iff succeeds
removed ← false
for each \(x\) in Domain\(\{X_j\}\) do
    if no value \(y\) in Domain\(\{X_j\}\) allows \((x, y)\) to satisfy the constraint \(X_i \iff X_j\)
    then delete \(x\) from Domain\(\{X_j\}\); removed ← true
return removed
```

- Runtime: \(O(n^2d^3)\), can be reduced to \(O(n^2d^2)\)
- … but detecting all possible future problems is NP-hard – why?

---

**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 \(O((n/c)(d^c))\), linear in \(n\)
  - E.g., \(n = 80, d = 2, c = 20\)
  - \(2^{80} = 4\) billion years at 10 million nodes/sec
  - \((4)(2^{20}) = 0.4\) seconds at 10 million nodes/sec
Theorem: if the constraint graph has no loops, the CSP can be solved in $O(n d^2)$ time (next slide)
- 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.

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 $\text{RemoveInconsistent}(\text{Parent}(X_i), X_i)$
For $i = 1 : n$, assign $X_i$ consistently with $\text{Parent}(X_i)$

Runtime: $O(n d^2)$
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 $O\left( (d^c)(n-c) d^2 \right)$, very fast for small $c$

Iterative Algorithms for CSPs

- Greedy and local methods 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., 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: $h(n) = \text{number of attacks}$

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

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

```plaintext
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
  schedule, a mapping from time to “temperature”
  local variables: current, a node
                  next, a node
                  T, a “temperature” controlling prob. of downward steps
  current ← MAKE-NODE(INITIAL-STATE[problem])
  for t ← 1 to ∞ do
    T ← schedule[t]
    if T = 0 then return current
    next ← a randomly selected successor of current
    ∆E ← VALUE[next] - VALUE[current]
    if ∆E > 0 then current ← next
    else current ← next only with probability e^(-∆E/T)
```
Simulated Annealing

- Theoretical guarantee:
  - Stationary distribution: \( p(x) \propto e^{\frac{E(x)}{kT}} \)
  - 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

\[
\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial y_1} & \frac{\partial f}{\partial x_2} & \frac{\partial f}{\partial y_2} & \frac{\partial f}{\partial x_3} & \frac{\partial f}{\partial y_3} \end{pmatrix}
\]

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

Image from vias.org