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)

Announcements

- Project 1.2 is up (Single-Agent Pacman)
  - Critical update: make sure you have the most recent version!
- Reminder: you are allowed to work with a partner!
- Change to John’s section: M 3-4pm now in 4 Evans

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?

Today

- Local search
- Robot motion planning

Hill Climbing Diagram

- Random restarts?
- Random sideways steps?
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 $h(n) = \text{total number of violated constraints}$

Example: 4-Queens

- States: 4 queens in 4 columns ($4^4 = 256$ states)
- Operators: move queen in column
- Goal test: no attacks
- Evaluation: $h(n) = \text{number of attacks}$

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

Simulated Annealing

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

```javascript
function SIMULATED-ANNEALING(problem, schedule)
    returns a solution state
    local variables: current, a node
    next, a node
    $T$, a "temperature" controlling prob. of downhill steps
    current = MAKE-NODE(INITIAL-STATE[problem])
    for $t = 1$ to $\infty$
        $T$ = schedule()
        if $T = 0$ then return current
        next = a randomly selected successor of current
        $\Delta E = \text{VALUE}[next] - \text{VALUE}[current]$
        if $\Delta E < 0$ then current = next
        else current = next with probability $e^{\Delta E / T}$
```

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

  Greedy Search

  - 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

Robotics Tasks

- Motion planning (today)
  - How to move from A to B
  - Known obstacles
  - Offline planning
- Localization (later)
  - Where exactly am I?
  - Known map
  - Ongoing localization (why?)
- Mapping (much later)
  - What’s the world like?
  - Exploration / discovery
  - SLAM: simultaneous localization and mapping

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
    \[
    \nabla f = \left( \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3} \right)
    \]
    \[
    x \leftarrow x + \alpha \nabla f(x)
    \]
Mobile Robots
- High-level objectives: move around obstacles, etc
- Low-level: fine motor control to achieve motion
- Why is this hard?

Manipulator Robots
- High-level goals: reconfigure environment
- Low-level: move from configuration A to B (point-to-point motion)
  - Why is this already hard?
- Also: compliant motion

Sensors and Effectors
- Sensors vs. Percepts
  - Agent programs receive percepts
  - Agent bodies have sensors
  - Includes proprioceptive sensors
  - Real world: sensors break, give noisy answers, miscalibrate, etc.
- Effectors vs. Actuators
  - Agent programs have actuators (control lines)
  - Agent bodies have effectors (gears and motors)
  - Real-world: wheels slip, motors fail, etc.

Degrees of Freedom
- The degrees of freedom are the numbers required to specify a robot’s configuration
- Positional DOFs:
  - (x, y, z) of free-flying robot
  - direction robot is facing
- Effector DOFs
  - Arm angle
  - Wing position
- Static state: robot shape and position
- Dynamic state: derivatives of static DOFs (why have these?)

Example
- How many DOFs?
  - What are the natural coordinates for specifying the robot’s configuration?
  - These are the configuration space coordinates
  - What are the natural coordinates for specifying the effector tip’s position?
  - These are the work space coordinates

Example
- How many DOFs?
  - How does this compare to your arm?
  - How many are required for arbitrary positioning of end-effector?
**Holonomicity**

- **Holonomic robots control all their DOFs (e.g., manipulator arms)**
  - Easier to control
  - Harder to build

- **Non-holonomic robots do not directly control all DOFs (e.g., a car)**

---

**Configuration Space**

- **Workspace:**
  - The world’s (x, y) system
  - Obstacles specified here

- **Configuration space**
  - The robot’s state
  - Planning happens here
  - Obstacles can be projected to here

- **Configuration space**
  - $r \in [1, 4]$ 

---

**Obstacles in C-Space**

- **What / where are the obstacles?**
- **Remaining space is free space**

---

**Kinematics**

- **Kinematics**
  - The mapping from configurations to workspace coordinates
  - Generally involves some trigonometry
  - Usually pretty easy

- **Inverse Kinematics**
  - The inverse: effector positions to configurations
  - Usually non-unique (why?)

**Forward kinematics**

\[ x = r \cos(\alpha) \]
\[ y = r \sin(\alpha) \]
You very quickly get into issues of topology:
- Point robot in 3D: $\mathbb{R}^3$
- Directional robot with fixed position in 3D: $\text{SO}(3)$
- Two rotational-jointed robot in 2D: $\text{S}_1 \times \text{S}_1$
- For the present purposes, we'll just ignore these issues
- In practice, you have to deal with it properly

Example: 2D Polygons

Example: Rotation

Example: A Less Simple Arm

Summary
- Degrees of freedom
- Legal robot configurations form configuration space
- Obstacles have complex images in c-space

Motion as Search
- Motion planning as path-finding problem
  - Problem: configuration space is continuous
  - Problem: under-constrained motion
  - Problem: configuration space can be complex
Decomposition Methods

- Break c-space into discrete regions
- Solve as a discrete problem

Hierarchical Decomposition

- Actually used in practical systems

- But:
  - Not optimal
  - Not complete
  - Still hopeless above a small number of dimensions

Exact Decomposition?

- With polygon obstacles: decompose exactly
- Problems?
  - Doesn’t scale at all
  - Doesn’t work with complex, curved obstacles

Skeletonization Methods

- Decomposition methods turn configuration space into a grid

- Skeletonization methods turn it into a set of points, with preset linear paths between them

Approximate Decomposition

- Break c-space into a grid
  - Search (A*, etc)
  - What can go wrong?
    - If no path found, can subdivide and repeat
- Problems?
  - Still scales poorly
  - Incomplete*
  - Wiggly paths

Visibility Graphs

- Shortest paths:
  - No obstacles: straight line
  - Otherwise: will go from vertex to vertex
  - Fairly obvious, but somewhat awkward to prove
- Visibility methods:
  - All free vertex-to-vertex lines (visibility graph)
  - Searching using, e.g., A*
  - Can be done in O(n^3) easily, O(n^2log(n)) less easily
- Problems?
  - Bang, screech!
  - Not robust to control errors
  - Wrong kind of optimality?
Voronoi Decomposition

- Voronoi regions: points colored by closest obstacle

- Voronoi diagram: borders between regions
  - Can be calculated efficiently for points (and polygons) in 2D
  - In higher dimensions, some approximation methods

Voronoi Decomposition

- Algorithm:
  - Compute the Voronoi diagram of the configuration space
  - Compute shortest path (line) from start to closest point on Voronoi diagram
  - Compute shortest path (line) from goal to closest point on Voronoi diagram
  - Compute shortest path from start to goal along Voronoi diagram

- Problems:
  - Hard over 2D, hard with complex obstacles
  - Can do weird things:

Probabilistic Roadmaps

- Idea: just pick random points as nodes in a visibility graph

- This gives probabilistic roadmaps
  - Very successful in practice
  - Lets you add points where you need them
  - If insufficient points, incomplete, or weird paths

Potential Field Methods

- So far: implicit preference for short paths
- Rational agent should balance distance with risk!
- Idea: introduce cost for being close to an obstacle
- Can do this with discrete methods (how?)
- Usually most natural with continuous methods

Potential Fields

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