More on HW 2 (due Jan 26)

• Again, it must be in Python 2.7.
• For the A* algorithm, you will need an Open* list and a Closed list.
• States should have
  – the coordinates of the point
  – the g-value cost of the path from init to here
  – the h-value estimate of cost to goal
  – the parent state
  – (optional) list of successors

*Note: Python will have a fit if you call it Open.
Start Point
Goal Point
How many Rectangles.
Rectangle coordinates are given clockwise.
More Difficult Example

Has 4 known solutions with approximately the same cost.

You will just find 1 least cost solution and print the whole path with states and cumulative costs.
In Addition

• You must design your own custom data set and run your program on it, too. You will turn in a picture of your data set, similar to the pictures we give you of ours.

• Turn in commented source code, input and output from all 3 data sets, and your picture.
Beyond Classical Search

• Chapter 3 covered problems that considered the whole search space and produced a sequence of actions leading to a goal.

• Chapter 4 covers techniques (some developed outside of AI) that don’t try to cover the whole space and only the goal state, not the steps, are important.

• The techniques of Chapter 4 tend to use much less memory and are not guaranteed to find an optimal solution.
More Search Methods

• Local Search
  – Hill Climbing
  – Simulated Annealing
  – Beam Search
  – Genetic Search
• Local Search in Continuous Spaces
• Searching with Nondeterministic Actions
• Online Search (agent is executing actions)
Local Search Algorithms and Optimization Problems

- **Complete state** formulation
  - For example, for the 8 queens problem, all 8 queens are on the board and need to be moved around to get to a goal state
- Equivalent to **optimization problems** often found in science and engineering
- Start somewhere and try to get to the solution from there
- **Local search** around the current state to decide where to go next
Pose Estimation Example

• Given a geometric model of a 3D object and a 2D image of the object.
• Determine the position and orientation of the object wrt the camera that snapped the image.

• State \((x, y, z, \theta_x, \theta_y, \theta_z)\)
Gradient

- What is the gradient of a function?
- In 1D. Function $f(x)$. Gradient $f'(x)$, the derivative.
- In 2D. Function $f(x,y)$. Gradient $(\partial_x, \partial_y)$
- e.g. $f(x) = x^2$. $f'(x) = ?$
Hill Climbing

“Gradient ascent”

Often used for numerical optimization problems.

How does it work?

In continuous space, the gradient tells you the direction in which to move uphill.

Note: solutions shown here as max not min.
Numeric Example

- Normal distribution with 0 mean and 1 SD
- \( f(x) = c e^{-1/2}x^2 \)
- \( f'(x) = -x c e^{-1/2}x^2 \)
- \( f'(1) \) comes out negative, i.e. move backward
- \( f'(-1) \) comes out positive, i.e. move forward.
Steepest-Ascent Hill Climbing

- \textit{current} $\leftarrow$ start node
- loop do
  - \textit{neighbor} $\leftarrow$ a highest-valued successor of \textit{current}
  - if \textit{neighbor}.Value $\leq$ \textit{current}.Value then return \textit{current}.State
  - \textit{current} $\leftarrow$ \textit{neighbor}
- end loop

At each step, the current node is replaced by the best (highest-valued) neighbor.

This is sometimes called greedy local search.
Hill Climbing Search

What if current had a value of 12?
Hill Climbing Problems

Local maxima

Plateaus

Diagonal ridges

What is it sensitive to?
Does it have any advantages?
Solving the Problems

• Allow backtracking (What happens to complexity?)

• Stochastic hill climbing: choose at random from uphill moves, using steepness for a probability

• Random restarts: “If at first you don’t succeed, try, try again.”

• Several moves in each of several directions, then test

• Jump to a different part of the search space
Simulated Annealing

• Variant of hill climbing *(so up is good)*

• Tries to **explore** enough of the search space **early on**, so that the final solution is less sensitive to the start state

• May make some **downhill moves** before finding a good way to move uphill.
Simulated Annealing

- Comes from the physical process of annealing in which substances are raised to high energy levels (melted) and then cooled to solid state.

\[ p = e^{-\frac{\Delta E}{kT}} \]

where \( \Delta E \) is the positive change in energy level, \( T \) is the temperature, and \( k \) is Boltzmann’s constant.
Simulated Annealing

• At the beginning, the temperature is high.
• As the temperature becomes lower
  – $kT$ becomes lower
  – $\Delta E/kT$ gets bigger
  – $(-\Delta E/kT)$ gets smaller
  – $e^{(-\Delta E/kT)}$ gets smaller
• As the process continues, the probability of a downhill move gets smaller and smaller.
For Simulated Annealing

• $\Delta E$ represents the change in the value of the objective function.

• Since the physical relationships no longer apply, drop $k$. So $p = e^{-\Delta E/T}$

• We need an annealing schedule, which is a sequence of values of $T$: $T_0, T_1, T_2, ...$
Simulated Annealing Algorithm

- \( current \leftarrow \) start node;

- for each \( T \) on the schedule /* need a schedule */
  
  - \( next \leftarrow \) randomly selected successor of \( current \)
  
  - evaluate next; it it’s a goal, return it

  - \( \Delta E \leftarrow next.Value - current.Value \) /* already negated */
  
  - if \( \Delta E > 0 \)
    
    - then \( current \leftarrow next \) /* better than current */
    
    - else \( current \leftarrow next \) with probability \( e^{-(\Delta E/T)} \)

How would you do this probabilistic selection?
Probabilistic Selection

• Select *next* with probability $p$

• Generate a random number

• If it’s $\leq p$, select *next*
Simulated Annealing Properties

• At a fixed “temperature” $T$, state occupation probability reaches the Boltzmann distribution

$$p(x) = \alpha e^{(E(x)/kT)}$$

• If $T$ is decreased slowly enough (very slowly), the procedure will reach the best state.

• Slowly enough has proven too slow for some researchers who have developed alternate schedules.
Simulated Annealing Schedules

• Acceptance criterion and cooling schedule

\[
\begin{align*}
\text{if } (\text{delta} \geq 0) & \text{ accept} \\
\text{else if } (\text{random} < e^{\frac{\text{delta}}{\text{Temp}}}) & \text{ accept, else reject } / * 0 \leq \text{random} \leq 1 /*
\end{align*}
\]

Initially temperature is very high (most bad moves accepted)
Temp slowly goes to 0, with multiple moves attempted at each temperature
Final runs with temp=0 (always reject bad moves) greedily “quench” the system
Simulated Annealing Applications

- Basic Problems
  - Traveling salesman
  - Graph partitioning
  - Matching problems
  - Graph coloring
  - Scheduling
- Engineering
  - VLSI design
    - Placement
    - Routing
    - Array logic minimization
    - Layout
  - Facilities layout
  - Image processing
  - Code design in information theory
Local Beam Search

- Keeps more previous states in memory
  - Simulated annealing just kept one previous state in memory.
  - This search keeps k states in memory.
  - randomly generate k initial states
  - if any state is a goal, terminate
  - else, generate all successors and select best k
  - repeat
Local Beam Search
Coming next: Genetic Algorithms, which are motivated by human genetics. How do you search a very large search space in a fitness oriented way?