Today’s Class

• Iterative improvement methods
  – Hill climbing
  – Simulated annealing
  – Local beam search
• Genetic algorithms
• Online search

These approaches start with an initial guess at the solution and gradually improve until it is one.
Hill climbing on a surface of states

Height Defined by Evaluation Function
Hill-climbing search

- Looks one step ahead to determine if any successor is better than the current state; if there is, move to the best successor.
- Rule:
  
  If there exists a successor $s$ for the current state $n$ such that
  
  - $h(s) < h(n)$ and
  - $h(s) \leq h(t)$ for all the successors $t$ of $n$,

  then move from $n$ to $s$. Otherwise, halt at $n$.

- Similar to Greedy search in that it uses $h()$, but does not allow backtracking or jumping to an alternative path since it doesn’t “remember” where it has been.

- Corresponds to Beam search with a beam width of 1 (i.e., the maximum size of the nodes list is 1).

- Not complete since the search will terminate at "local minima," "plateaus," and "ridges."
Hill climbing example

\[ f(n) = -\text{(number of tiles out of place)} \]
Exploring the Landscape

- **Local Maxima**: peaks that aren’t the highest point in the space.

- **Plateaus**: the space has a broad flat region that gives the search algorithm no direction (random walk).

- **Ridges**: flat like a plateau, but with drop-offs to the sides; steps to the North, East, South and West may go down, but a step to the NW may go up.

Image from: http://classes.yale.edu/fractals/CA/GA/Fitness/Fitness.html
Drawbacks of hill climbing

• Problems: local maxima, plateaus, ridges
• Remedies:
  – Random restart: keep restarting the search from random locations until a goal is found.
  – Problem reformulation: reformulate the search space to eliminate these problematic features
• Some problem spaces are great for hill climbing and others are terrible.
Example of a local optimum

\[
\begin{array}{c c c}
1 & 2 & 5 \\
8 & 7 & 4 \\
6 & 3 \\
\end{array}
\]

\(f = -6\)

\[
\begin{array}{c c c}
1 & 2 & 5 \\
7 & 4 & \\
8 & 6 & 3 \\
\end{array}
\]

\(f = -7\)

start

move up

move right

\[
\begin{array}{c c c}
1 & 2 & 5 \\
8 & 7 & 4 \\
6 & 3 \\
\end{array}
\]

f = -7

\[
\begin{array}{c c c}
1 & 2 & 3 \\
8 & 4 & \\
7 & 6 & 5 \\
\end{array}
\]

\(f = 0\)

goal

f = -(manhattan distance)
Gradient ascent / descent

- Gradient descent procedure for finding the $\arg_x \min f(x)$
  - choose initial $x_0$ randomly
  - repeat
    - $x_{i+1} \leftarrow x_i - \eta f'(x_i)$
    - until the sequence $x_0, x_1, \ldots, x_i, x_{i+1}$ converges
- Step size $\eta$ (eta) is small (perhaps 0.1 or 0.05)
Gradient methods vs. Newton’s method

• A reminder of Newton’s method from Calculus:
  \[ x_{i+1} \leftarrow x_i - \eta f'(x_i) / f''(x_i) \]

• Newton’s method uses 2\textsuperscript{nd} order information (the second derivative, or, curvature) to take a more direct route to the minimum.

• The second-order information is more expensive to compute, but converges quicker.

Contour lines of a function
Gradient descent (green)
Newton’s method (red)

Simulated annealing

• Simulated annealing (SA) exploits an analogy between the way in which a metal cools and freezes into a minimum-energy crystalline structure (the annealing process) and the search for a minimum [or maximum] in a more general system.

• SA can avoid becoming trapped at local minima.

• SA uses a random search that accepts changes that increase objective function $f$, as well as some that decrease it.

• SA uses a control parameter $T$, which by analogy with the original application is known as the system “temperature.”

• $T$ starts out high and gradually decreases toward 0.
Simulated annealing (cont.)

- A “bad” move from $A$ to $B$ is accepted with a probability
  \[ P(\text{move}_{A \to B}) = e^{(f(B) - f(A)) / T} \]

- The higher the temperature, the more likely it is that a bad move can be made.
- As $T$ tends to zero, this probability tends to zero, and SA becomes more like hill climbing.
- If $T$ is lowered slowly enough, SA is complete and admissible.
The simulated annealing algorithm

function SIMULATED-ANNEALING(problem, schedule) returns a solution state

inputs: problem, a problem
        schedule, a mapping from time to “temperature”

static: current, a node
        next, a node
        T, a “temperature” controlling the probability 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
    ΔE ← VALUE[next] − VALUE[current]
    if ΔE > 0 then current ← next
    else current ← next only with probability $e^{ΔE/T}$
Local beam search

• Begin with $k$ random states
• Generate all successors of these states
• Keep the $k$ best states

• Stochastic beam search: Probability of keeping a state is a function of its heuristic value
Genetic algorithms

• Similar to stochastic beam search
• Start with $k$ random states (the initial population)
• New states are generated by “mutating” a single state or “reproducing” (combining via crossover) two parent states (selected according to their fitness)
• Encoding used for the “genome” of an individual strongly affects the behavior of the search

• Genetic algorithms / genetic programming are a large and active area of research
In-Class Paper Discussion

Class Exercise:
Local Search for Map/Graph Coloring
Online search

- Interleave computation and action (search some, act some)
- Exploration: Can’t infer outcomes of actions; must actually perform them to learn what will happen

- Competitive ratio = Path cost found* / Path cost that could be found**
  * On average, or in an adversarial scenario (worst case)
  ** If the agent knew the nature of the space, and could use offline search

- Relatively easy if actions are reversible (ONLINE-DFS-AGENT)
- LRTA* (Learning Real-Time A*): Update $h(s)$ (in state table) based on experience
- More about these issues when we get to the chapters on Logic and Learning!
**Summary: Informed search**

- **Best-first search** is general search where the minimum-cost nodes (according to some measure) are expanded first.

- **Greedy search** uses minimal estimated cost $h(n)$ to the goal state as measure. This reduces the search time, but the algorithm is neither complete nor optimal.

- **A* search** combines uniform-cost search and greedy search: $f(n) = g(n) + h(n)$. A* handles state repetitions and $h(n)$ never overestimates.
  - A* is complete and optimal, but space complexity is high.
  - The time complexity depends on the quality of the heuristic function.
  - IDA* and SMA* reduce the memory requirements of A*.

- **Hill-climbing algorithms** keep only a single state in memory, but can get stuck on local optima.

- **Simulated annealing** escapes local optima, and is complete and optimal given a “long enough” cooling schedule.

- **Genetic algorithms** can search a large space by modeling biological evolution.

- **Online search** algorithms are useful in state spaces with partial/no information.