Local Search
AI Class 6 (Ch. 4.1-4.2)

Based on slides by Dr. Marie desJardin. Some material also adapted from slides by Dr. Matuszek @ Villanova University, which are based on Hwee Tou Ng at Berkeley, which are based on Russell at Berkeley. Some diagrams are based on AIMA.
Bookkeeping

- HW 1 due last night
  - Grades within 1.5 weeks (hopefully sooner)
  - Discussions after grading

- HW 2 out tonight 11:59
  - Due 10/3, 11:59pm
Today’s Class

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

“If the path to the goal does not matter... [we can use] a single current node and move to neighbors of that node.”

– R&N pg. 121
Admissibility

- Admissibility is a property of **heuristics**
  - They are *optimistic* – think goal is closer than it is
  - (Or, exactly right)
- Admissible algorithms can be pretty bad!
- Is $h(n)$: “1 kilometer” admissible?
- Using admissible heuristics guarantees that the first solution found will be optimal, **for some algorithms** (A*).
Admissiblility and Optimality

• Intuitively:
  • When A* finds a path of length $k$, it has already tried every other path which can have length $\leq k$
  • Because all frontier nodes have been sorted in ascending order of $f(n) = g(n) + h(n)$

• Does an admissible heuristic guarantee optimality for greedy search?
  • Reminder: $f(n) = h(n)$, always choose node “nearest” goal
  • No sorting beyond that
Local Search Algorithms

• Sometimes the path to the goal is irrelevant
  • Goal state itself is the solution
  • $\exists$ an **objective function** to evaluate states

• In such cases, we can use local search algorithms

• Keep a single “current” state, try to improve it
Local Search Algorithms

- Sometimes the path to the goal is irrelevant
  - Goal state itself is the solution
  - There exists an **objective function** to evaluate states

- State space = set of “complete” configurations
  - That is, all elements of a solution are present
  - Find configuration satisfying constraints
  - Example?

- In such cases, we can use local search algorithms
- Keep a single “current” state, try to improve it
What Is This?
Iterative Improvement Search

- Start with an initial guess
- Gradually improve it until it is legal or optimal
- Some examples:
  - Hill climbing
  - Simulated annealing
  - Constraint satisfaction
Hill Climbing on State Surface

- Concept: trying to reach the “highest” (most desirable) point (state)
- “Height” Defined by Evaluation Function
Hill Climbing Search

• If there exists a successor \( s \) for the current state \( n \) such that
  • \( h(s) < h(n) \)
  • \( h(s) \leq h(t) \) for all the successors \( t \) of \( n \),
then move from \( n \) to \( s \). Otherwise, halt at \( n \).

• Look one step ahead to determine if any successor is “better” than current state
  • If so, move to the best successor

• A kind of Greedy search in that it uses \( h \)
  • But, does not allow backtracking or jumping to an alternative path
  • Doesn’t “remember” where it has been.

• Not complete
  • Search will terminate at local minima, plateaux, ridges.
Hill Climbing Example

start

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

\[h = -4\]

-5

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

\[h = -3\]

-5

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

\[h = -3\]

-3

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

\[h = -3\]

-4

f(n) = -(number of tiles out of place)

goal

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

\[h = 0\]

-2

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

\[h = -1\]

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

\[h = -2\]
Exploring the Landscape

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

- **Plateaus:**
  - 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.
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; others are terrible
Example of a Local Optimum

\[ f = -(\text{manhattan distance}) \]

\[ f = -6 \]

\[ f = -7 \]

\[ f = 0 \]
Some Extensions of Hill Climbing

- Simulated Annealing
  - Escape local maxima by allowing some “bad” moves but gradually decreasing their frequency

- Local Beam Search
  - Keep track of $k$ states rather than just one
  - At each iteration:
    - All successors of the $k$ states are generated and evaluated
    - Best $k$ are chosen for the next iteration
Some Extensions of Hill Climbing

- Stochastic Beam Search
  - Chooses semi-randomly from “uphill” possibilities
  - “Steeper” moves have a higher probability of being chosen

- Random-Restart Climbing
  - Can actually be applied to any form of search
  - Pick random starting points until one leads to a solution

- Genetic Algorithms
  - Each successor is generated from two predecessor (parent) states
Gradient Ascent / Descent

- Gradient descent procedure for finding the arg \( \min f(x) \)
  - 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 (~0.1–0.05)
- Good for differentiable, continuous spaces

Images from http://en.wikipedia.org/wiki/Gradient_descent
Gradient Methods vs. Newton’s Method

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

- 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 more quickly.

Simulated Annealing

- Simulated annealing (SA): analogy between the way metal cools into a minimum-energy crystalline structure and the search for a minimum generally
  - In very hot metal, molecules can move fairly freely
  - But, they are slightly less likely to move out of a stable structure
  - As you slowly cool the metal, more molecules are “trapped” in place
- Conceptually: Escape local maxima by allowing some “bad” (locally counterproductive) moves but gradually decreasing their frequency
Simulated Annealing (II)

- Can avoid becoming trapped at local minima.
- Uses a random local search that:
  - Accepts changes that increase objective function $f$
  - As well as some that decrease it
- Uses a control parameter $T$
  - By analogy with the original application
  - Is known as the system “temperature”
- $T$ starts out high and gradually decreases toward 0
Simulated Annealing (III)

- $f(s)$ represents the quality of state $n$ (high is good)
- A “bad” move from A to B is accepted with a probability
  \[ P(\text{move}_{A \rightarrow B}) \approx e^{(f(B) - f(A)) / T} \]
  - (Note that $f(B) - f(A)$ will be negative, so bad moves always have a relatively probability less than one. Good moves, for which $f(B) - f(A)$ is positive, have a relative probability greater than one.)

- Temperature
  - 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.
Visualizing SA Probabilities

\[-1,1\] ratio = 7.0286876
\[p(\text{neg}) = 0.1422741\]

\[-1,1\] ratio = 49.402449
\[p(\text{neg}) = 0.0202419\]

\[-1,1\] ratio = 298267566
\[p(\text{neg}) = 3.398 \times 10^{-9}\]
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
  - $k$, instead of one, current state(s)
- 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
  - More likely to keep “better” successors
Genetic Algorithms

• The Idea:
  • New states are generated by “mutating” a single state or “reproducing” (somehow combining) two parent states
  • Selected according to their **fitness**

• Similar to stochastic beam search

• Start with $k$ random states (the **initial population**)
  • 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
Class Exercise:
Local Search for N-Queens

(more on constraint satisfaction heuristics next time...)

<table>
<thead>
<tr>
<th>Q</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Q</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Q</td>
</tr>
<tr>
<td>Q</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Q</td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Q</td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Q</td>
<td></td>
</tr>
</tbody>
</table>
Tabu Search

- Problem: Hill climbing can get stuck on local maxima
- Solution: Maintain a list of \( k \) previously visited states, and prevent the search from revisiting them
- Why not always do this?
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

- LRTA* (Learning Real-Time A*): Update $h(s)$ (in state table) based on experience

- More about online search and nondeterministic actions next time…
Summary: Informed Search

- **Best-first search** is general search where the minimum-cost nodes are expanded first.

- **Greedy search** uses minimal estimated cost $h(n)$ to the goal state as measure

- Reduces the search time, but 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.
  - 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.