Artificial Intelligence Programming

Local Search

Cindi Thompson

Department of Computer Science

University of San Francisco

Overview

- Local Search - When is it useful?
- Hill-climbing search
- Simulated Annealing
- Genetic Algorithms

Local Search

- So far, the algorithms we've looked at store the entire path from initial state to goal state.
- This leads to memory/space issues on large problems.
- For some problems, path information is essential
  - Route finding
  - Rubik's Cube
  - 8-puzzle
  - The solution *is* the sequence of actions to take.
- We know what the goal state is, but not how to reach it.

Local Search

- For other sorts of problems, we may not care what the sequence of actions is.
  - Finding the optimal (or satisfactory) solution is what's important.
  - Scheduling
  - VLSI layout
  - Cryptography
  - Function optimization
  - Protein folding, gene sequencing
- The solution is an assignment of values to variables that maximizes some objective function.
- In these cases, we can safely discard at least some of the path information.

Local Search

- A search algorithm that uses only the current state (as opposed to path information) is called a *local search* algorithm.
- Advantages:
  - Constant memory requirements
  - Can find solutions in incredibly large spaces.
- Disadvantages:
  - Hard to guarantee optimality; we might only find a local optimum
  - Lack of memory may cause us to revisit states or oscillate.

Search Landscape

- Local search is often useful for optimization problems
- “Find parameters such that o(x) is maximized/minimized”
- This is a search problem, where the state space is the combination of value assignments to parameters.
- If there are n parameters, we can imagine an $n + 1$ dimensional space, where the first n dimensions are the parameters of the function, and the $n + 1$th dimension is the *objective function*.
- We call this space a *search landscape*
  - Optima are on hills
  - Valleys are poor solutions.
  - (reverse this to minimize $o(x)$)

Optimization example

A farmer has 2400 ft of fencing and wants to fence off a rectangular field that borders a straight river. He needs no fence along the river. What are the dimensions of the field that has the largest area?

Optimization example

2400 ft of fencing for a rectangular field bordering a straight river. What are the dimensions leading to the largest area?

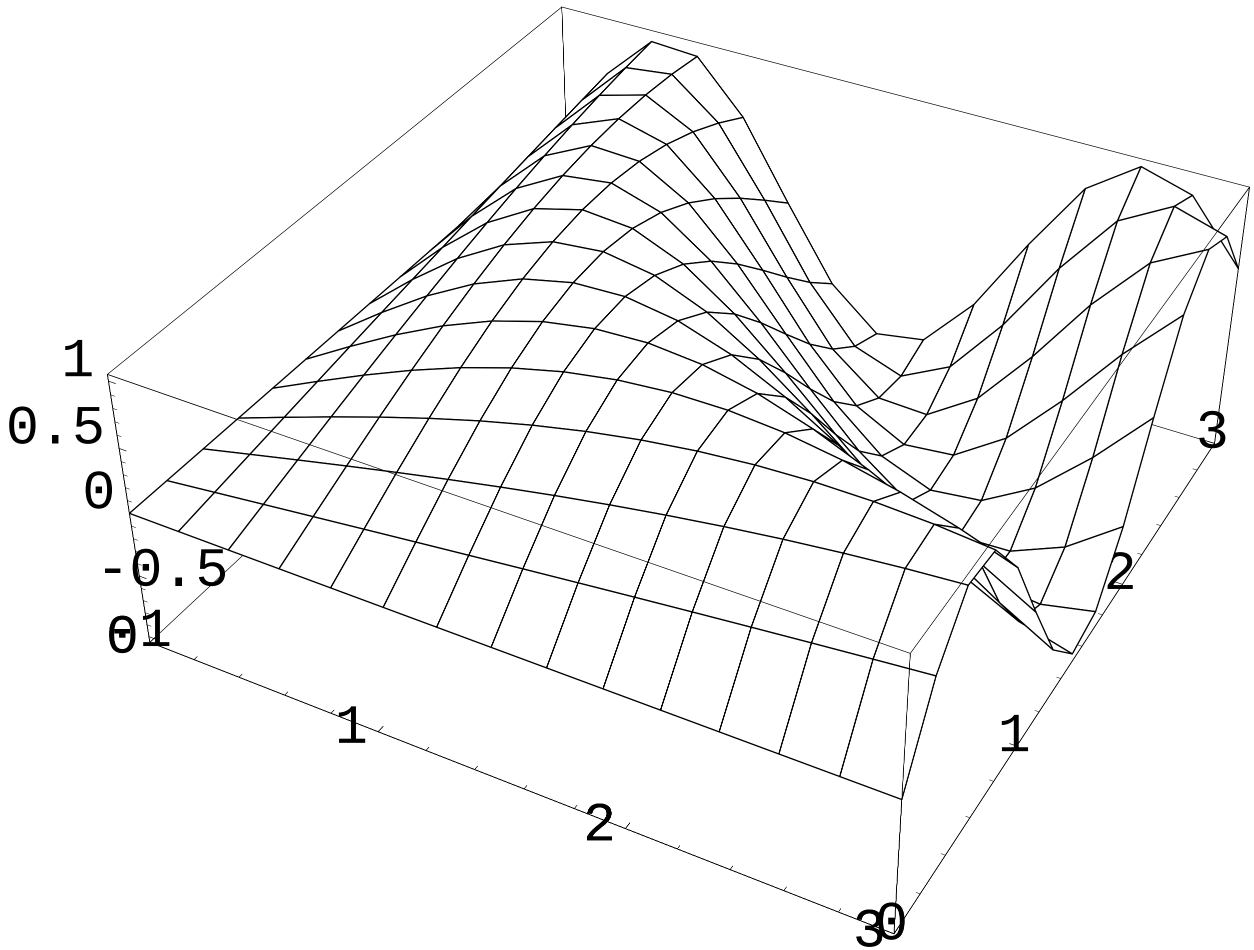
- Maximize: $A = xy$
- Constraint: $2x + y = 2400$

Search Landscape

A one-dimensional landscape

Search Landscape

A two-dimensional landscape:



(beyond 2 dimensions, they're tough to draw)

Search landscapes

- Landscapes turn out to be a very useful metaphor for local search algorithms.
- Lets us visualize ‘climbing’ up a hill (or descending a valley).
- Gives us a way of differentiating easy problems from hard problems.
  - Easy: few peaks, smooth surfaces, no ridges/plateaus
  - Hard: many peaks, jagged or discontinuous surfaces, plateaus.

Hill-climbing search

- The simplest form of local search is hill-climbing search.
- Very simple: at any point, look at your "successors" (neighbors) and move in the direction of the greatest positive change.
- Very similar to greedy search
  - Pick the choice that myopically looks best.
- Very little memory required.
- Will get stuck in local optima.
- Plateaus can cause the algorithm to wander aimlessly.

Hill-climbing example

N-Queens

- Each state is represented by an n-unit vector
- $V[i] =$ Position of queen (1-n) in column i
- Optimization function, o, is number of conflicts
- Try to minimize o

Local search in Calculus

- Find roots of an equation $f(x) = 0$, f differentiable.
- Guess an x_1, find $f(x_1)$, $f'(x_1)$
- Use the tangent line to $f(x_1)$ (slope = $f'(x_1)$) to pick x_2.
- Repeat. $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_1)}$
- This is a hill-climbing search.
- Works great on smooth functions.

Improving hill-climbing

- Hill-climbing can be appealing
  - Simple to code
  - Requires little memory
  - We may not have a better approach.
- How to make it better?
- Stochastic hill-climbing - pick randomly from uphill moves
  - Weight probability by degree of slope

Improving hill-climbing

Random-restart hill-climbing

- Run until an optimum is reached
- Randomly choose a new initial state
- Run again.
- After n iterations, keep best solution.
  - If we have a guess as to the number of optima, we can choose an n.

Simulated Annealing

- Hill-climbing's weakness is that it never moves "downhill"
- Like greedy search, it can't "back up".
- Simulated annealing is an attempt to overcome this.
- "Bad" actions are occasionally chosen to move out of a local optimum.

Simulated Annealing

- Based on analogies to crystal formation.
- When a metal cools, lattices form as molecules fit into place.
- By reheating and recooling, a harder metal is formed
  - Small undoing leads to better solution.
  - Minimize the “energy” in the system
- Similarly, small steps away from the solution can help hill-climbing escape local optima.

Simulated Annealing

```
T = initial
s = initial-state
while (s != goal)
   ch = successor-fn(s)
   c = select-random-child(ch)
   delta-E = o(c) - o(s)
   if c is better than s
      s = c
   else
      s = c with probability proportional to k(T, delta-E)
   update T
```

- What is T?
- What is k?

Simulated Annealing

- What we want to do is make “mistakes” more frequently early in the search and more rarely later in the search.
- We’ll use T to parameterize this.
- T stands for temperature.
- Two questions:
  - What’s the probability function with respect to T?
  - How does T change over time?

Boltzmann distribution

- The probability of accepting a mistake is governed by a *Boltzmann distribution*
- Let s be the current state, c be the child considered, and o the function to optimize.
- P(c) = $exp(-\frac{o(c)-o(s)}{T})$
- Consider boundary conditions:
  - T very high: most fractions near 0, P(c) near 1.
  - T low: P(c) depends on $o(c) - o(s)$
    - $o(c) - o(s)$ near 0, then P(c) near 1.
    - $o(c) - o(s)$ large, then P(c) is small.

Boltzmann gives us a way of weighting the probability of accepting a “mistake” by its quality.

Cooling schedule

- The function for changing T is called a cooling schedule.
- The most commonly used schedules are:
  - Linear: $T_{new} = T_{old} - dt$
  - Proportional: $T_{new} = c * T_{old}, c < 1$

Simulated Annealing

- Simulated Annealing is complete and optimal as long as T is lowered “slowly enough”
- Can be very effective in domains with many optima.
- Simple addition to a hill-climbing algorithm.
- Weakness - selecting a good cooling schedule.
- No problem knowledge used in search. (weak method)