CS 331: Artificial Intelligence
Local Search 1

Tough real-world problems

Suppose you had to solve VLSI layout problems (minimize distance between components, unused space, etc.)…

Or schedule airlines…

Or schedule workers with specific skill sets to do tasks that have resource and ordering constraints

What do these problems have in common?

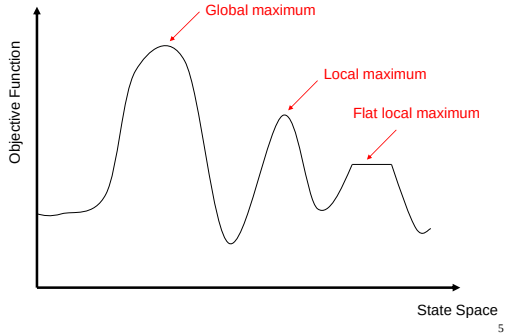
- These problems are unlike the search problems from last class:
  - The path to the goal is irelevant -- all you care about is the final configuration
  - These are often optimization problems in which you find the best state according to an objective function
- These problems are examples of local search problems

Local Search Problems

- Given a set of states S = $\{X_1, \ldots, X_m\}$
- And an objective function Eval(X_i) that returns the "goodness" of a state
- Find the state X* that maximizes the objective function

Note: Sometimes Eval(X_i) is a cost function instead of an objective function. In this case, we want to find the state X* that minimizes the cost function. We will deal with objective functions in these slides but it's easy to just flip the signs and think of cost functions.

1D State-Space Landscape



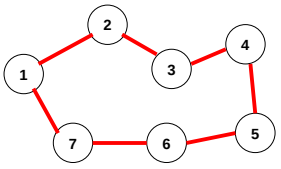
Why is this hard?

- Lots of states (sometimes an infinite number)
- Most of these problems are NP-complete
- Objective function might be expensive

But:

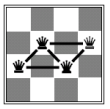
- Use very little memory (usually constant)
- Find reasonable (but usually not optimal) solutions in large or infinite state spaces

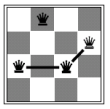
More examples
Traveling Salesman Problem



- State X = order of cities visited such that all cities are visited exactly once
- Eval(X) = length of the tour defined by state X
- Want to minimize the length of the tour

More Examples
n-queens


Eval(X) = 5


Eval(X) = 2


Eval(X) = 0

- State X = placement of the queens on the board
- Eval(X) = # of pairs of queens attacking each other
- Want to minimize Eval(X)

Local Search Algorithm Recipe

1. Start with initial configuration X
2. Evaluate its neighbors ie. the set of all states reachable in one move from X
3. Select one of its neighbor X*
4. Move to X* and repeat until the current configuration is satisfactory

How you define the neighborhood is important to the performance of the algorithm.

Which neighbor you select is also very important

Common stopping criteria:
- Run for specified # of iterations
- Run for specified time
- Run until you can't move uphill

Outline

1. Hill-climbing
2. Simulated Annealing
3. Beam Search
4. Genetic Algorithms

1. Hill-climbing

Hill-climbing (Intuitively)

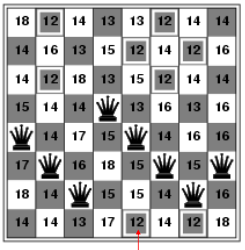
- "…resembles trying to find the top of Mount Everest in a thick fog while suffering from amnesia."
- Starting at initial state X, keep moving to the neighbor with the highest objective function value greater than X's.

Hillclimbing Pseudocode

X ← Initial configuration
Iterate:
  E ← Eval(X)
  N ← Neighbors(X)
  For each X_i in N
    E_i ← Eval(X_i)
  E* ← Highest E_i
  X* ← X_i with highest E_i
  If E* > E
    X ← X*
  Else
    Return X

13

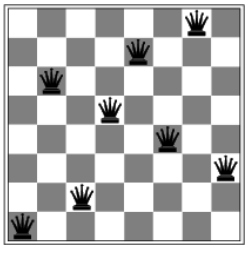
Example: 8-queens



- State: a board with 8 queens
- Successor function: move a single queen to any other square in same column
- Heuristic cost function (h): # of pairs of queens attacking each other

These numbers are the value of the heuristic cost function if you move the queen in that column to that square

Example: 8-queens



- Local minimum with h=1
- Note: at global minimum, h=0

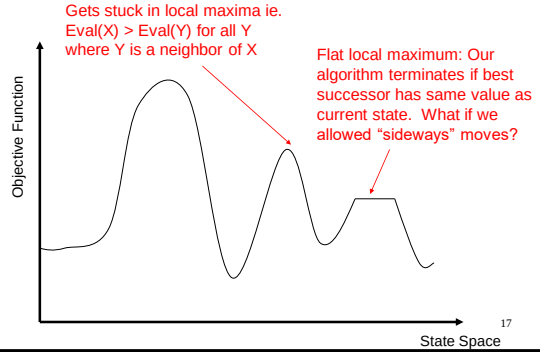
15

More on hill-climbing

- Hill-climbing also called greedy local search
- Greedy because it takes the best immediate move
- Greedy algorithms often perform quite well

16

Problems with Hill-climbing

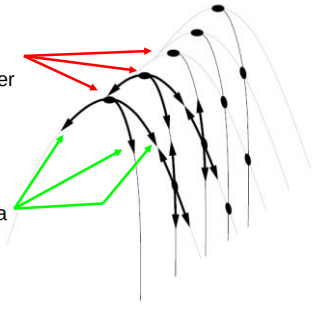


17

Problems with Hill-climbing (Ridges)

Sequence of local maxima not directly connected to each other

All available actions from each local maxima go downhill



Neighborhoods

- Recall that we said that defining the neighborhood correctly is critical to the performance of the algorithm
- Large neighborhood: Lots of evaluations to do at each state but better chance of finding a good maximum
- Small neighborhood: Fewer evaluations at each state but can potentially get stuck in more local maxima

19

Variants of Hill-climbing

- Stochastic hill climbing:
  - Chooses at random among the uphill moves
  - Probability of selection varies with steepness
- First-choice hill climbing:
  - Generates successors randomly until one is generated that is better than the current state
  - Good when state has many successors
- Random-restart hill-climbing
  - Good for dealing with local maxima
  - Conduct a series of hill-climbing searches from randomly generated initial states
  - Stop when a goal state is found (or until time runs out, in which case return the best state found so far)

Which variant?

- Depends on the state-space landscape (which is difficult to know a priori)
- Typical real-world problems have an exponential number of local maxima
- But hill-climbing still works reasonably well

21

Simulated Annealing

- Hill-climbing never makes a downhill move
- What if we added some random moves to hill-climbing to help it get out of local maxima?
- This is the motivation for simulated annealing

If you're curious, **annealing** refers to the process used to harden metals by heating them to a high temperature and then gradually cooling them

22

2. Simulated Annealing

23

Simulated Annealing Pseudocode

```
X ← Initial configuration
Iterate:
    E ← Eval(X)
    X' ← Randomly selected neighbor of X
    E' ← Eval(X')
    If E' ≥ E
        X ← X'
        E ← E'
    Else with probability p
        X ← X'
        E ← E'
```

24

Simulated Annealing Pseudocode

X ← Initial configuration
Iterate:
  E ← Eval(X)
  X' ← Randomly selected neighbor of X
  E' ← Eval(X')
  If E' ≥ E
    X ← X'
    E ← E'
  Else with probability p
    X ← X'
    E ← E'

This part is different from hillclimbing

Simulated Annealing Pseudocode

X ← Initial configuration
Iterate:
  E ← Eval(X)
  X' ← Randomly selected neighbor of X
  E' ← Eval(X')
  If E' ≥ E
    X ← X'
    E ← E'
  Else with probability p
    X ← X'
    E ← E'

How do you set p?

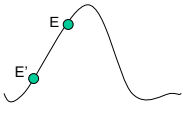
Setting p

- What if p is too low?
  - We don't make many downhill moves and we might not get out of many local maxima
- What if p is too high?
  - We may be making too many suboptimal moves
- Should p be constant?
  - We might be making too many random moves when we are near the global maximum

Setting p

- **Decrease p as iterations progress**
  - Accept more downhill moves early, accept fewer as search goes on
  - Intuition: as search progresses, we are moving towards more promising areas and quite likely toward a global maximum
- **Decrease p as E-E' increases**
  - Accept fewer downhill moves if slope is high
  - See next slide for intutition

Decreasing p as E-E' increases



E-E' is large: we are likely moving towards a sharp (interesting) maximum so don't move downhill too much

E-E' is small: we are likely moving towards a smooth (uninteresting) maximum so we want to escape this local maximum

Setting p

- Needs a temperature parameter T
- If E' ≤ E, accept the downhill move with probability p = $e^{-(E-E')/T}$
- Start with high temperature T, (more downhill moves allowed at the start)
- Decrease T gradually as iterations increase (less downhill moves allowed)
- Annealing schedule describes how T is decreased at each step

Complete Simulated Annealing Pseudocode

X ← Initial configuration
Iterate:
  Do K times:
    E ← Eval(X)
    X' ← Randomly selected neighbor of X
    E' ← Eval(X')
    If E' ≥ E
      X ← X'
      E ← E'
    Else with probability $p = e^{-(E-E')/T}$
      X ← X'
      E ← E'
  T = αT

Complete Simulated Annealing Pseudocode

X ← Initial configuration
Iterate:
  Do K times:
    E ← Eval(X)
    X' ← Randomly selected neighbor of X
    E' ← Eval(X')
    If E' ≥ E
      X ← X'
      E ← E'
    Else with probability $p = e^{-(E-E')/T}$
      X ← X'
      E ← E'
  T = αT

We keep the temperature fixed and we iterate K times

Exponential cooling schedule T(n) = α T(n-1) with 0 < α < 1.

Convergence

- If the schedule lowers T slowly enough, the algorithm will find a global optimum with probability approaching 1
- In practice, reaching the global optimum could take an enormous number of iterations

The fine print…

- Design of neighborhood is critical
- Lots of parameters to tweak eg. α, K, initial temperature
- Simulated annealing is usually better than hillclimbing if you can find the right parameters

What You Should Know

- Be able to formulate a problem as a local search problem
- Know the difference between local search and uninformed and informed search
- Know how hillclimbing works
- Know how simulated annealing works
- Know the pros and cons of both methods

Exercise

You have to move from your old apartment to your new one. You have the following:

- A list $L = \{a_1, a_2, \ldots, a_n\}$ of n items, each with a size $s(a_i) > 0$.
- There are M moving boxes available, each with a box capacity C (assume MC far exceeds the sum of the sizes of your items). You can put as many items into a box as long as the sum of their sizes does not exceed the box capacity C.

Your job is to pack your stuff into as few boxes as possible. Formulate this as a local search problem.

Exercise (continued)

- States?
- Neighborhood?
- Evaluation function?
- How to avoid local maxima?

37