EECS 391: Introduction to AI

Soumya Ray

Website: http://vorlon.case.edu/~sray/eecs391_sp12/index.html

Email: sray@case.edu

Office: Olin 516

Office hours: Tue 2:30-4:00 or by appointment

Announcements

- HW2/PA1 due next week

Today

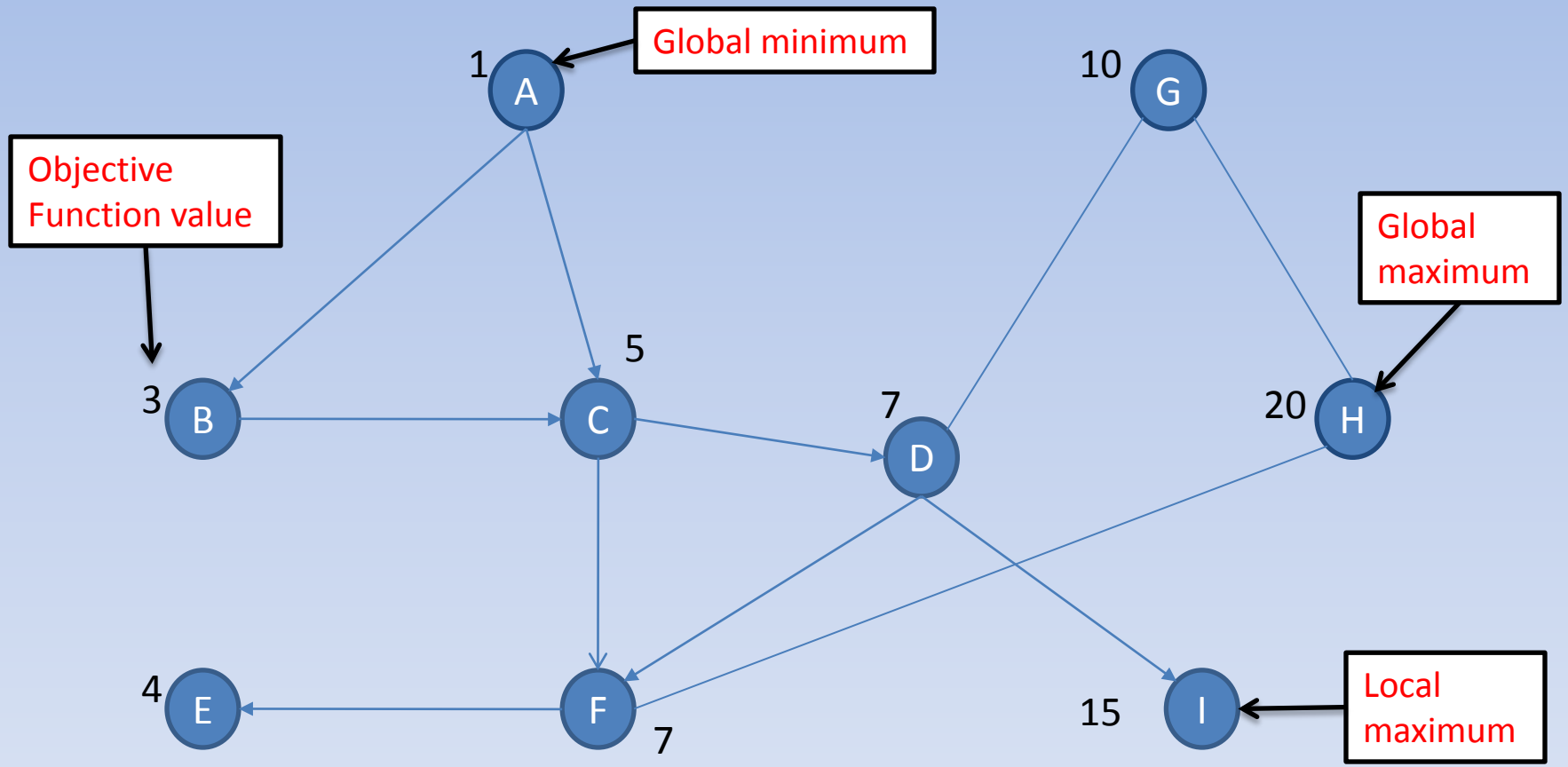
- Search for Optimization Problems (Chapter 4)
- Intro to Adversarial Search (Chapter 5)

Optimization Problems

- So far, we have used search to find optimal paths to a known goal ("goal-directed search")

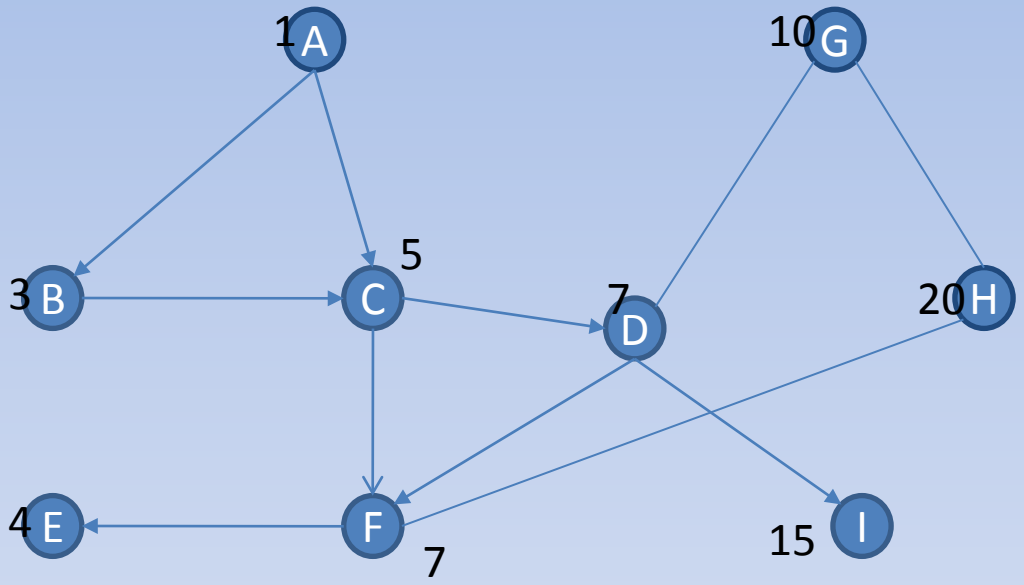
- Consider a space where each state is associated with a function value, and we want to find the state with the max or min value
  - This is an optimization problem
  - Search is useful here as well

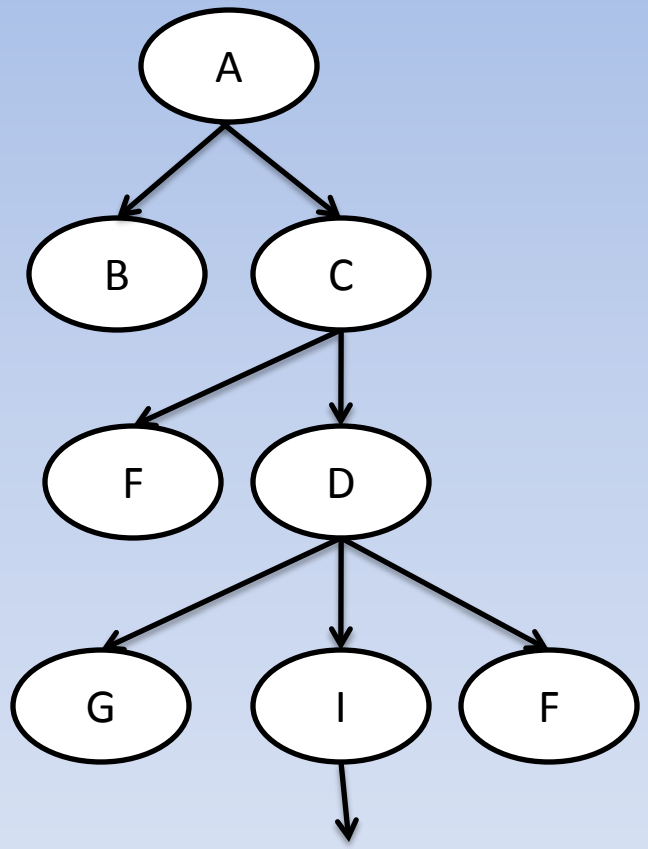
Example: Discrete Optimization



Plateau: Node and all neighbors have the same function value

Example: Hill Climbing





Return max value: 15

Random Restart Hill Climbing

- To fix the local optimum problem, we often iterate basic hill climbing in a loop
  - For constant number of trials
    - Choose random initial state (or use random tie breaking)
    - Run Hill Climbing
    - Record Solution
  - Return largest solution over all trials

Stochastic Search

- The previous algorithms are all deterministic
  - Keep going as long as value increases, then stop
  - Might get stuck, might need to cross “bad patch” to get to real solution

- Stochastic search methods add randomness to the search procedure
  - With some probability, allow “bad moves”
    - Move to a neighbor with worse function value

Simulated Annealing

- Combines Hill Climbing with a random walk in the search space
- Introduce a “Temperature” parameter, T
- Instead of always making “good” moves, algorithm can make “bad” moves, with probability governed by T

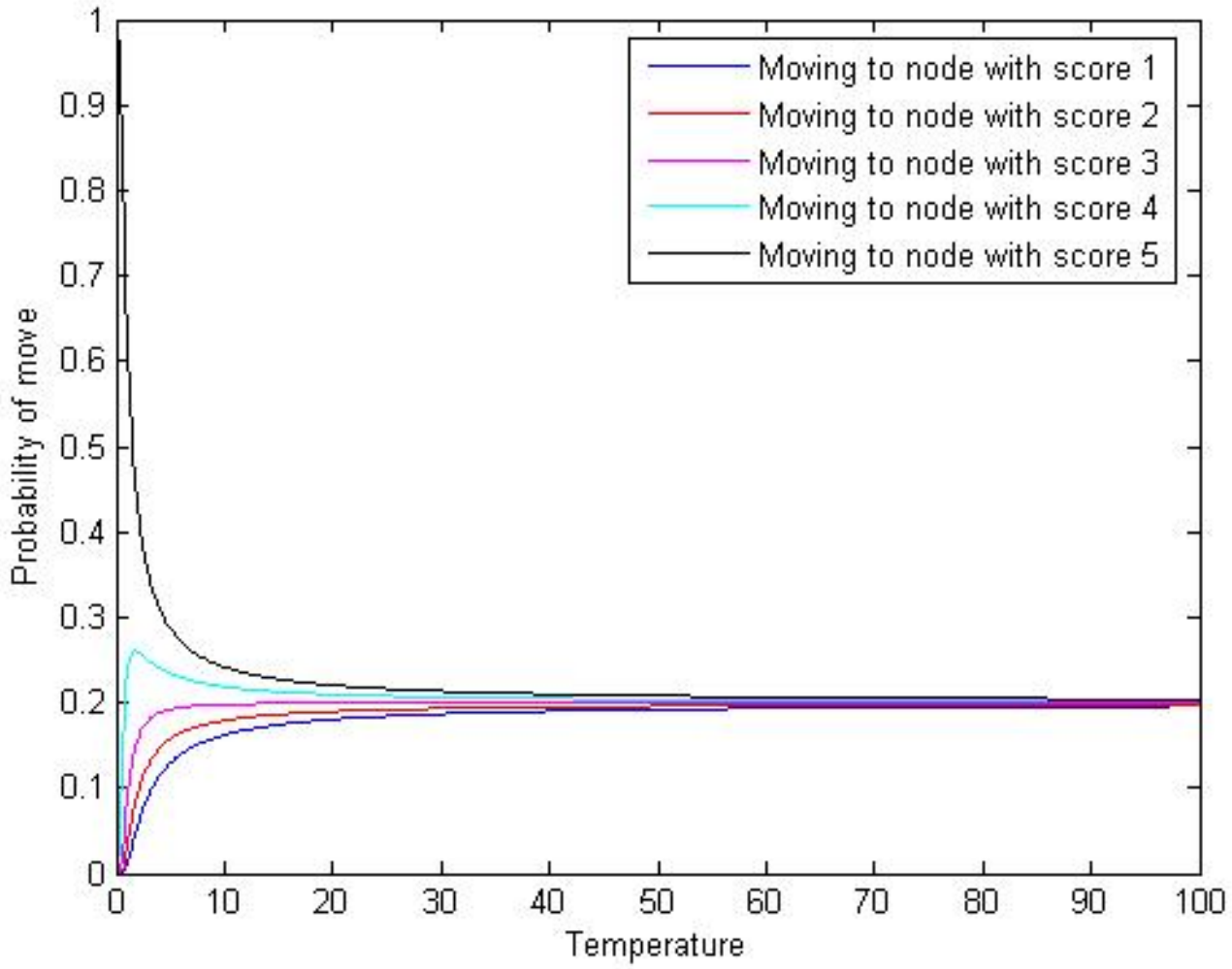
SA Move Selection

- Initialize T to a high value
- Suppose the current node has n neighbors with function values $f_1,\ldots, f_n$. Of these, suppose the max is f_{max}
- Move to neighbor i with probability:

$$\Pr(\text{move to } i^{th} \text{ neighbor}) \propto \exp\left(\frac{f_i - f_{\max}}{T}\right)$$

- On the next iteration, reduce the temperature T according to a "schedule"

Effect of Temperature

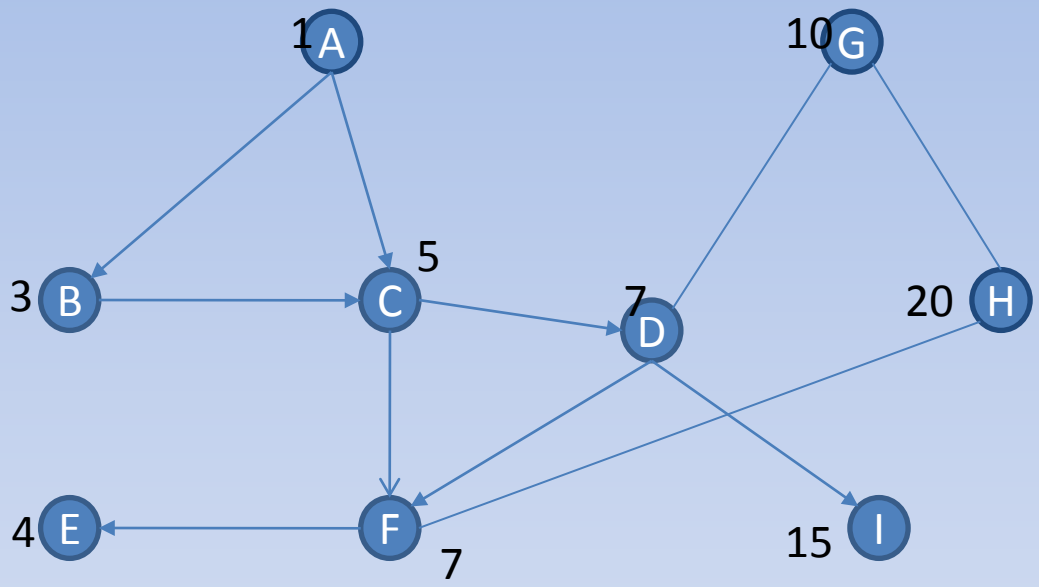


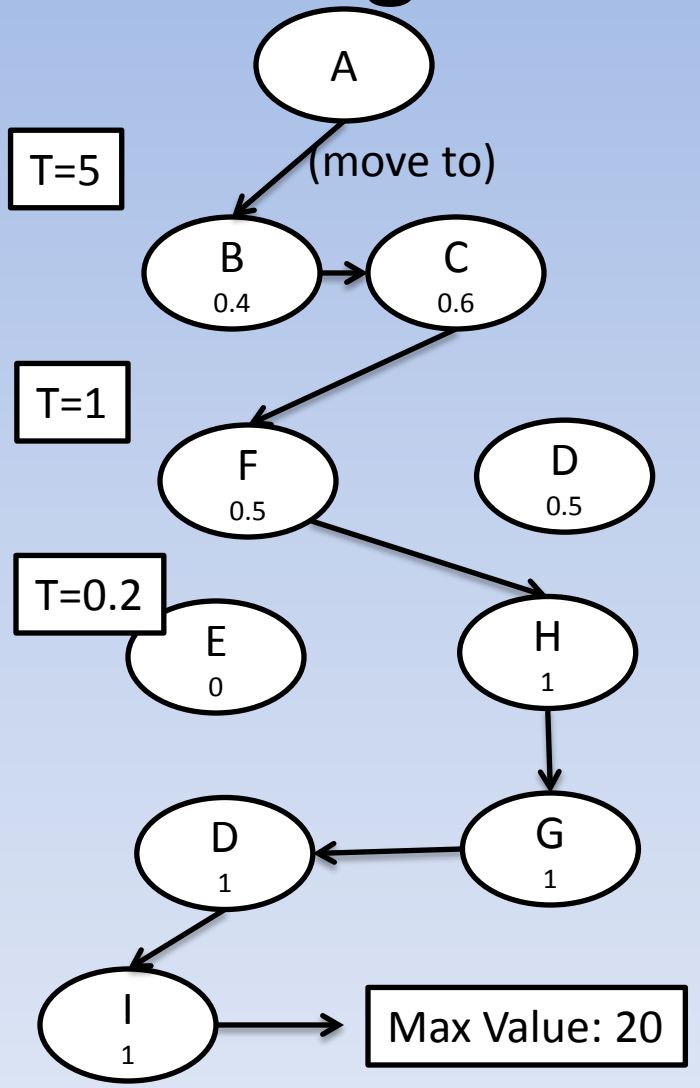
Suppose a node has 5 neighbors with function values 1, 2, 3, 4, 5. This graph shows the probability simulated annealing will move to each node as the temperature T varies. The probability is $exp((f_i-f_{max})/T)$, as on the previous slide, normalized to sum to 1.

Simulated Annealing Behavior

- When T is high, moves randomly
- When T is low, behaves like Hill Climbing (moves to node with max function value)
- Stopping Criterion: All neighbors visited

Example: Simulated Annealing





Expanding A: Pr(move to B)∝exp(-2/5)=0.67
Pr(move to C)∝exp(0)=1
Pr(move to B)=0.67/1.67=0.4
Pr(move to C)=1/1.67=0.6

Genetic Algorithms

- "Nature-inspired" stochastic search
  - Tries to mimic Darwinian evolution

- At each step, these algorithms maintain a "population" of states
  - The population "evolves" over time, with "more fit" (larger function value) members being preferred
  - After several steps, fittest state is returned

Representing States

States are represented by strings over a finite alphabet

	3	5
7	8	1
2	6	4

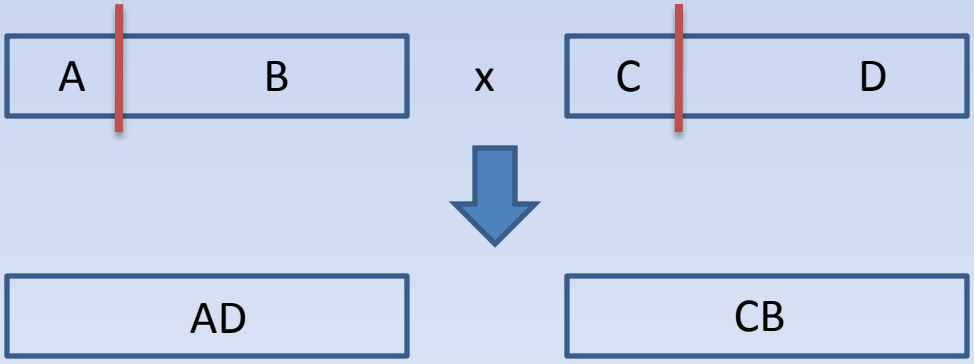
"B35781264"

Evolution in GAs

- At each step, the population of states "evolves"
- The initial population is ranked by the "fitness function" (the function to be optimized)
- States are chosen according to the ranking (more "fit" states chosen more often) and altered via mutation and crossover
  - These are the search operators in GAs

Mutation and Crossover

- Mutation: A single position in the string describing the chosen state is chosen and altered with some probability
- Crossover: Two states are chosen and a "crossover point" is marked. The crossover operator then produces two new states as follows:



Checking Legality

- Mutation and crossover, unless carefully defined, may lead to illegal states
  - Check and prune
  - E.g. mutation in 8-puzzle

Solution

- After some number of iterations, stop and pick the most fit state and return its value as solution

- Notice that the “standard” search operators are not used at all
  - Might create a problem if you also need some way to get from an initial state to the solution

Why does this work?

- Basically, it is a form of stochastic beam search

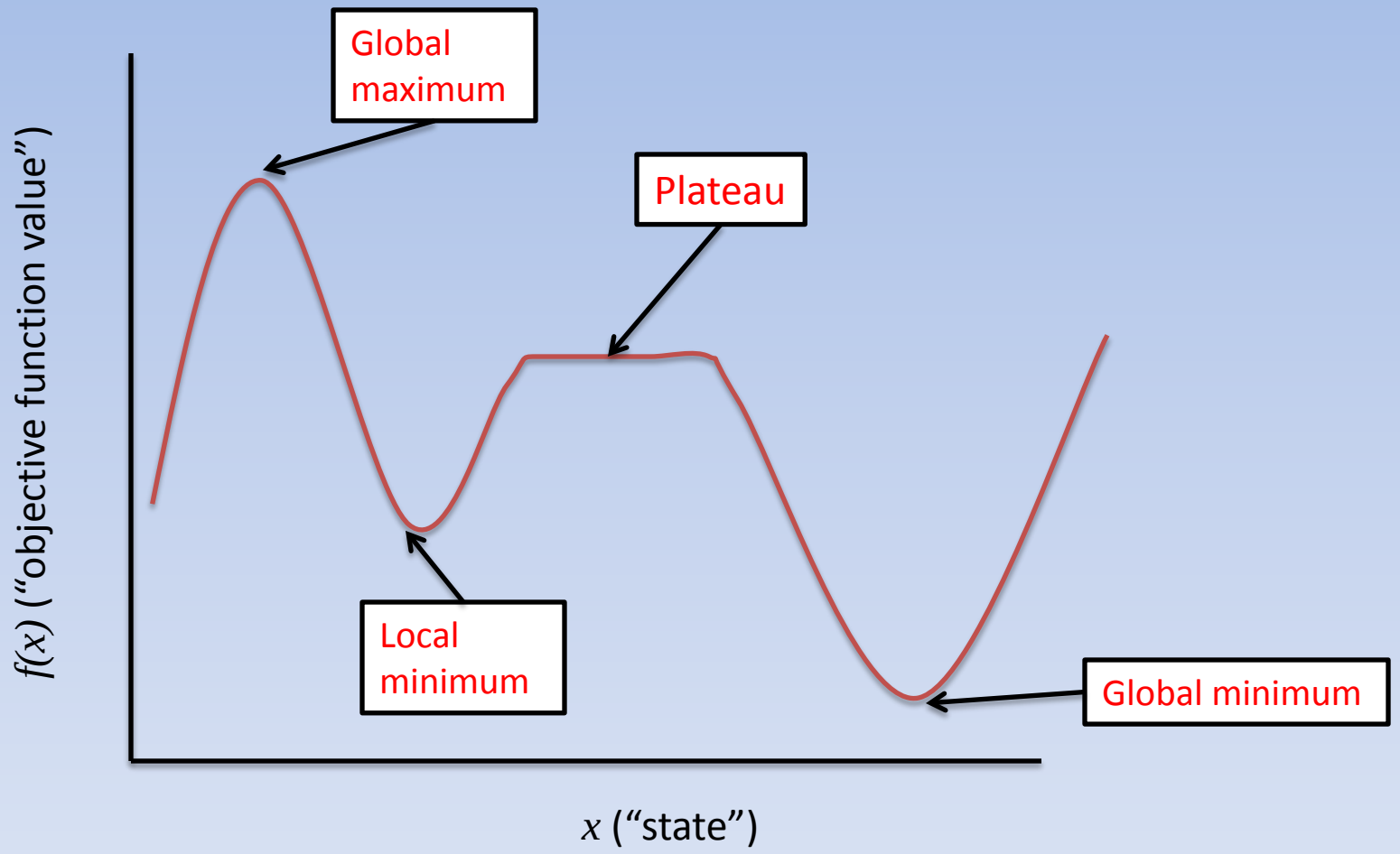
- Ideally, the fitness of the population grows over time, leading to optimum
  - But, also leads to problems with homogeneity, so ad hoc heuristics about “diversity” often used

Why does this work?

- Uses a more “structured” state representation, but does not reason with it

- Sometimes (often) it won’t work
  - Lots of parameters/random choices, hard to model effects/analyze behavior
  - But occasionally will surprise you

Continuous Optimization



Optimization in Continuous Spaces

- Previous methods do not work in continuous spaces
- But here, we can use tools from calculus
  - These can also be viewed as local search algorithms

Gradient Ascent/Descent

- Analog of Hill Climbing
- From the current state, move in the gradient direction (for maximization) or negative gradient (for minimization)

$$\mathbf{x}_{new} = \mathbf{x}_{old} + \alpha \nabla f_{\mathbf{x}_{old}}(\mathbf{x})$$

Stepsize

Function Gradient evaluated at $\mathbf{x}_{old}$

Newton-Raphson Method

- From the current state, take a Newton step:

$$\mathbf{x}_{new} = \mathbf{x}_{old} - \left[\nabla^2 f_{\mathbf{x}_{old}}(\mathbf{x})\right]^{-1} \nabla f_{\mathbf{x}_{old}}(\mathbf{x})$$

$$f(\mathbf{x}_{old} + u) = f(\mathbf{x}_{old}) + u^T \nabla f_{\mathbf{x}_{old}}(\mathbf{x}) + \frac{1}{2} u^T \nabla^2 f_{\mathbf{x}_{old}}(\mathbf{x}) u = g(u)$$

Newton Step

Set $\frac{\partial g}{\partial u} = 0,$ then

$$\nabla f_{\mathbf{x}_{old}}(\mathbf{x}) + \nabla^2 f_{\mathbf{x}_{old}}(\mathbf{x}) u = 0$$

and

$$u = -\left[\nabla^2 f_{\mathbf{x}_{old}}(\mathbf{x})\right]^{-1} \nabla f_{\mathbf{x}_{old}}(\mathbf{x})$$

Summary

- We learned about:
  - Search for Discrete Optimization Problems
    - Hill-climbing, Beam Search, Simulated Annealing
  - Search for Continuous Optimization Problems
    - Gradient Ascent/Descent, Newton-Raphson method

Adversarial Search

- Previous discussion assumed single agent

- Now consider *two competing* agents in deterministic turn-taking setting
  - Called a "game"
  - Generalizes to multiple agents

Games in AI

- Games have a long history in AI
  - States are conceptually easy to represent
  - All relevant state information is available to you (except opponent)
  - Few actions
  - Highly structured, precise rules, clear termination
  - Often suggested as “proof-of-intelligence”
- E.g. chess worked on by Turing, Shannon, Wiener among others

Games

- We consider zero-sum games of perfect information

- Perfect Information: Everything except the other agent's strategy is known

- Zero-sum: when one agent wins, the other loses (by the same amount)

Zero-sum games

- A and B are playing a game
- Let's define a utility function for each state of the game for A:

$$U_A(s) = \begin{cases} c \text{ if A wins in } s, c > 0 \\ -c \text{ if B wins in } s \\ 0 \text{ otherwise} \end{cases}$$

- Therefore, A will try to maximize $U_A(s)$ and B will try to minimize it
  - A is also called "MAX" and B is called "MIN"

Games as Search

- Initial state=initial configuration of pieces, cards etc
- Goal state=?
- Search Operators=?
- Cost=?
- Utility function

But…what about the other agent?