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

- HW1 solution online
- HW2 out later today

Today

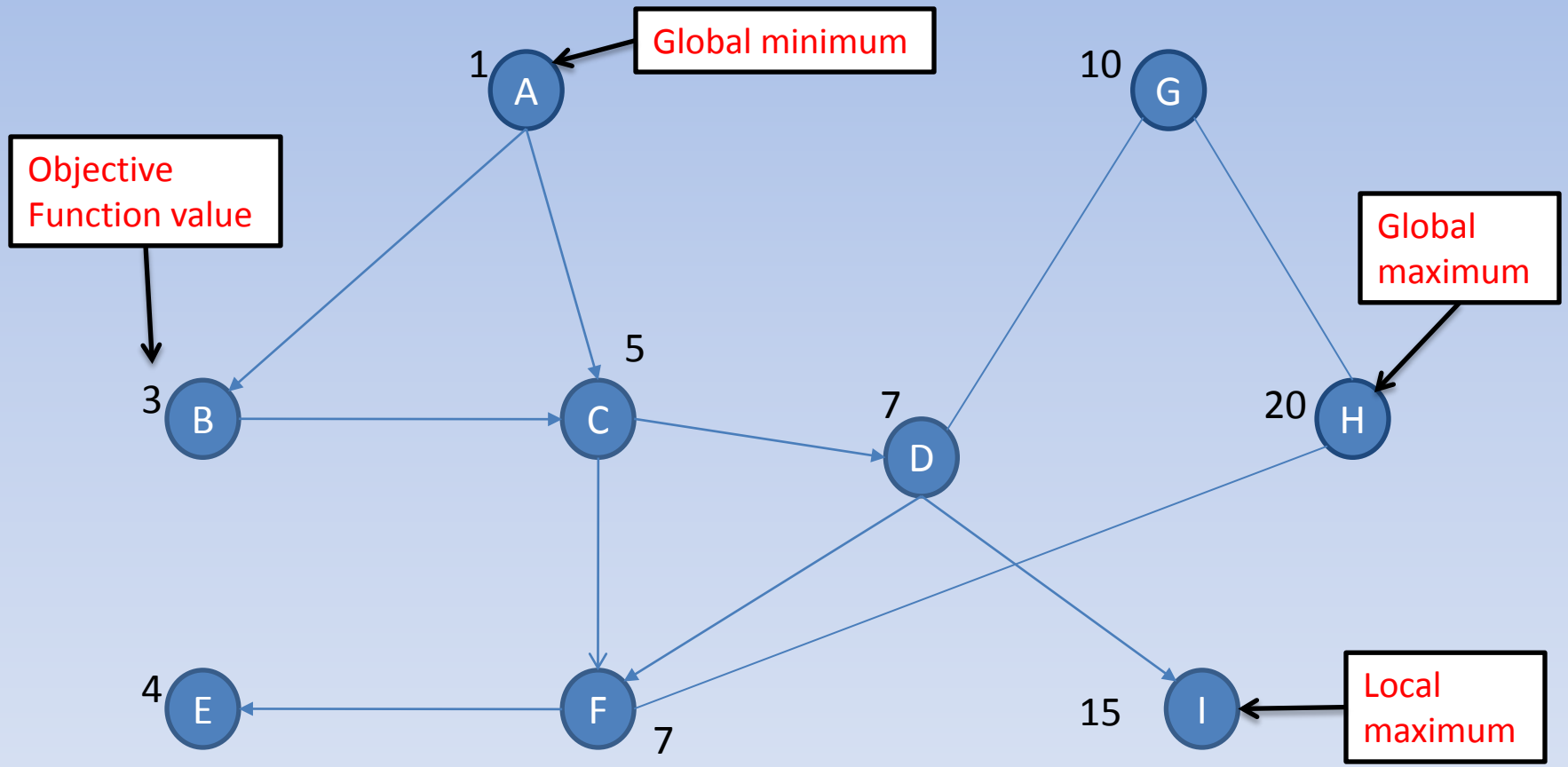
- Search for Optimization Problems (Chapter 4)

Optimization Problems

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

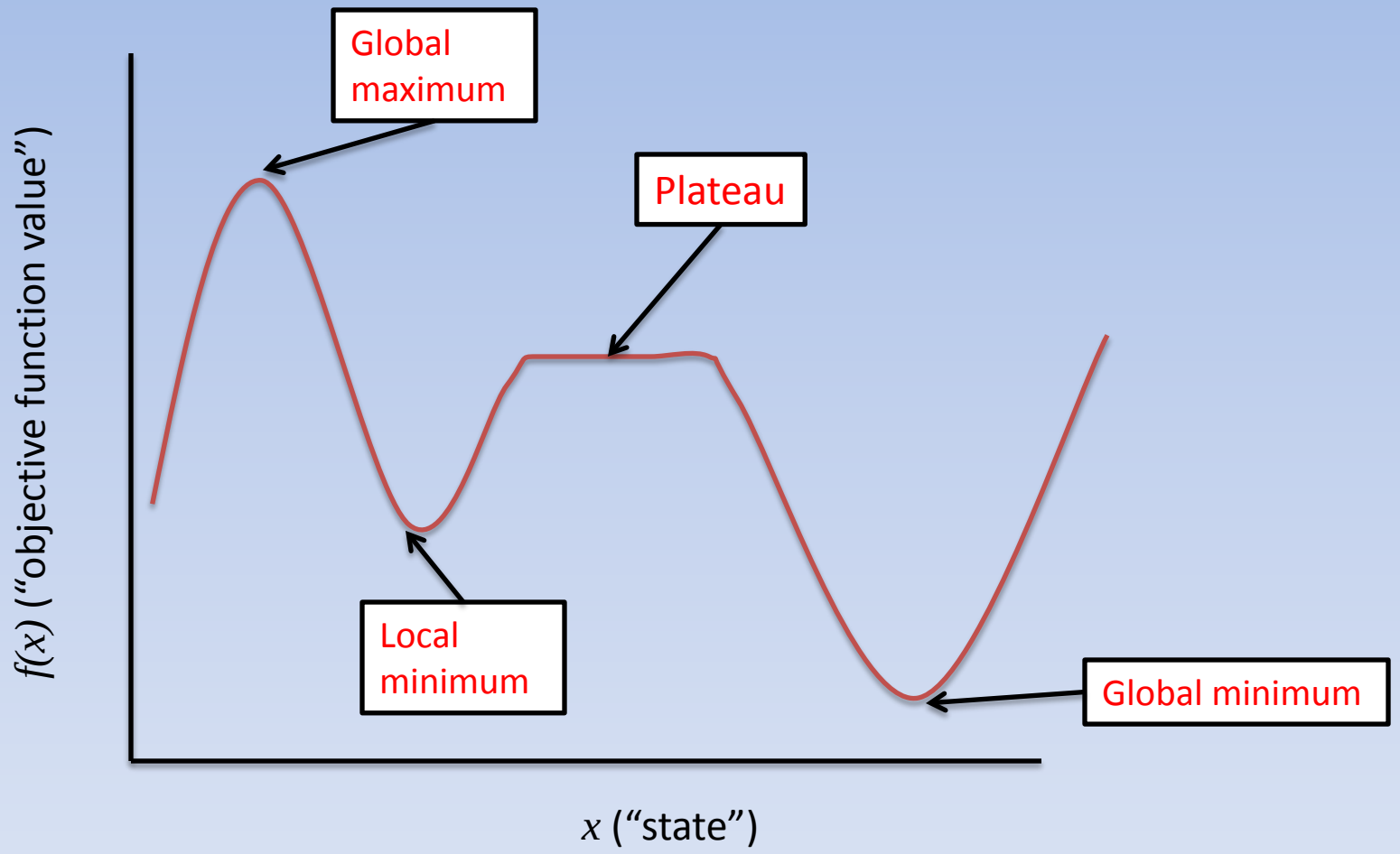
- Consider a space where each state is associated with a function value, and we want to find the state with the max or min value
  - This is an optimization problem
  - Search is useful here as well

Example: Discrete Optimization



Plateau: Node and all neighbors have the same function value

Example: Continuous Optimization



Differences from Goal-Directed Search

- In goal-directed search, we know the goal state ("State 253")
  - In optimization, we don't explicitly know the goal ("the state with the max value")
  - This is not a goal test
- In goal-directed search, the path cost matters (we try to minimize this)
  - In optimization, only the value of the state matters (not how we get there)
- Sometimes in optimization problems, the initial state won't be specified (agent can choose)

Example

- An agent playing chess could use search for optimization
  - Each state is associated with a function value representing win probability, maximize this
- But if it needs to win in 40 moves, the sequence of operators becomes important
  - Then use goal-directed search

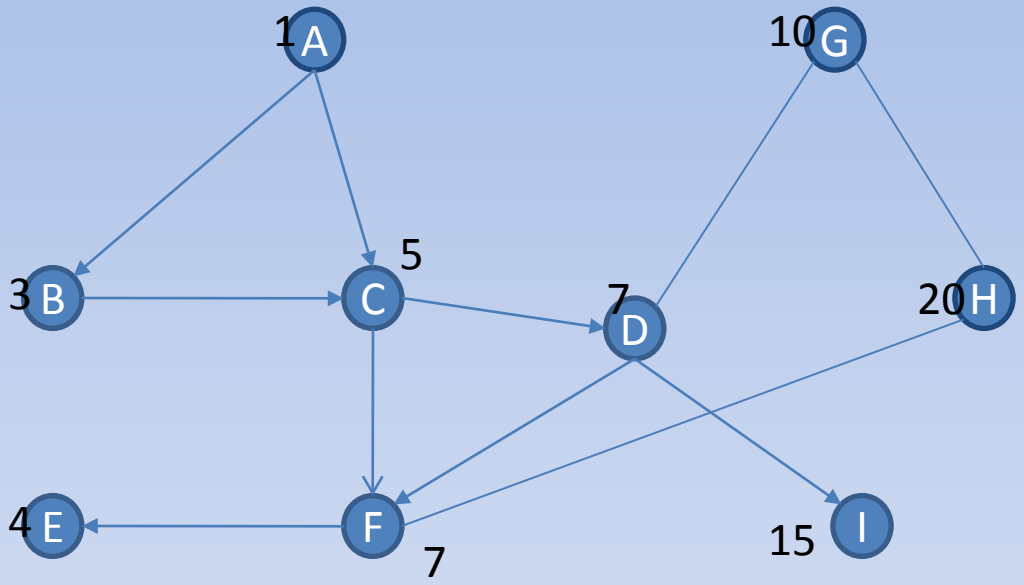
Search for Discrete Optimization

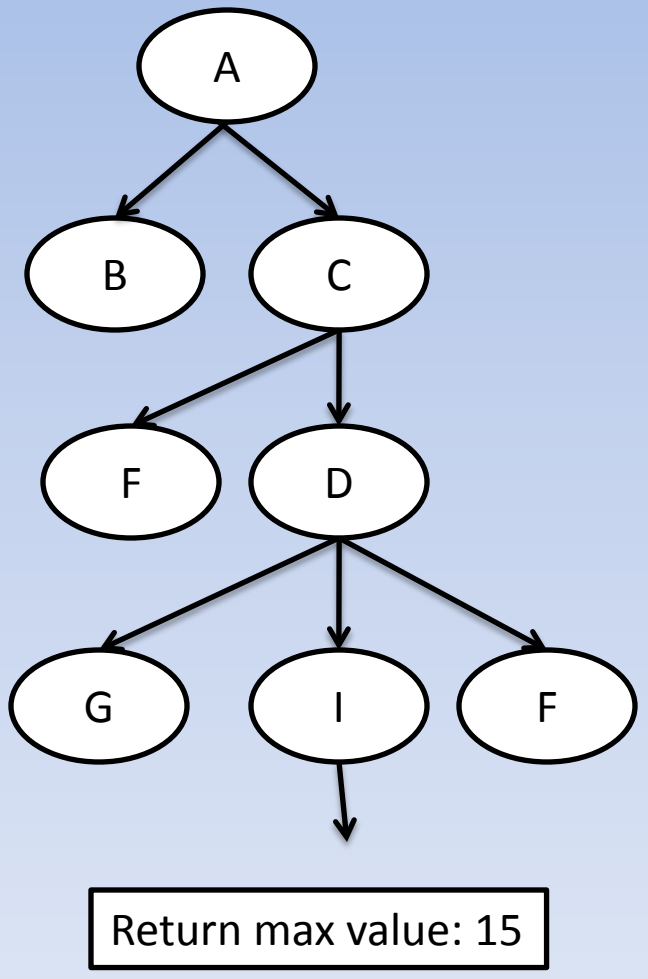
- For optimization, search algorithms do not need to track the path
  - These are called local search methods
- Basic local search steps (for maximization):
  - Start at initial state and apply search operators
  - If all neighbors have lower function value or are in closed list, stop
  - Else, select a neighbor and repeat
- Algorithms differ in how to select neighbors and in stopping condition

Hill Climbing Search

- Always select neighbor with largest function value greater than current
- Also called greedy local search
  - Very fast in most cases, needs very little memory
  - Can easily get stuck in local maximum/minimum
  - “Resembles trying to find the top of Mount Everest in a thick fog while suffering from amnesia”

Example





Return max value: 15

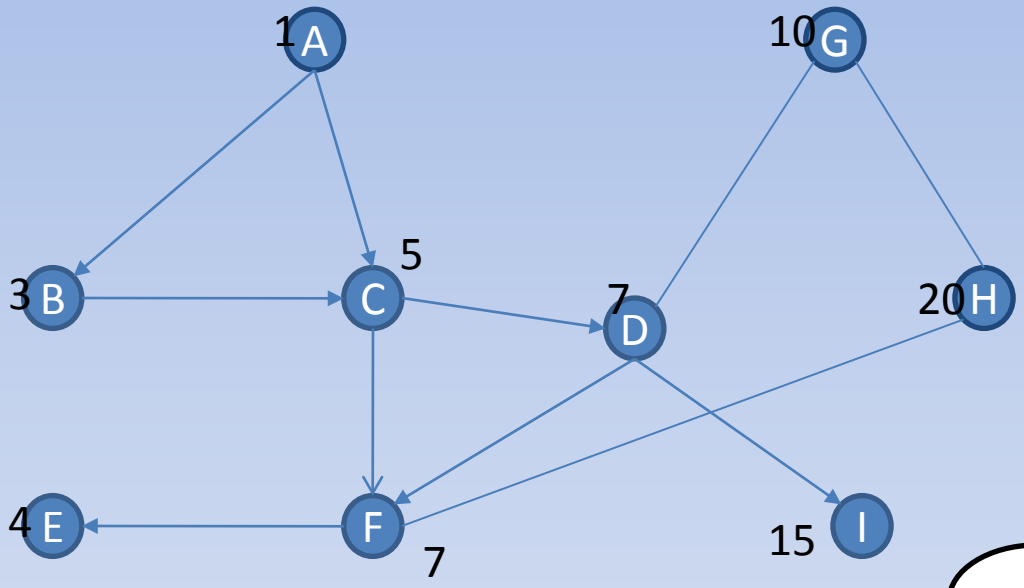
Random Restart Hill Climbing

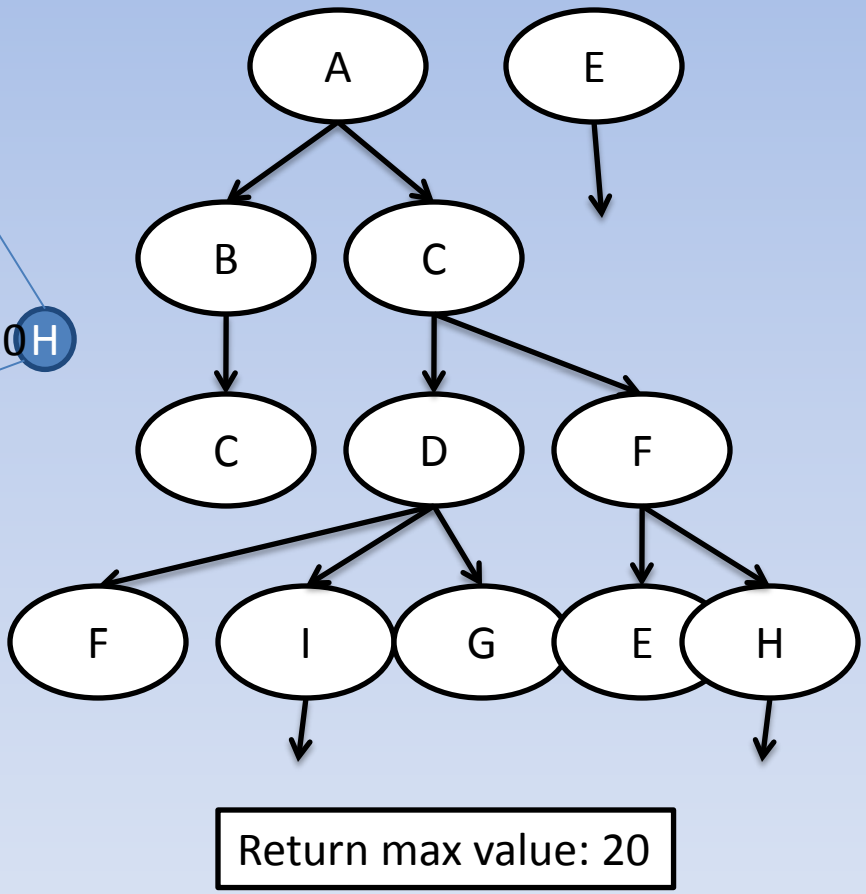
- To fix the local optimum problem, we often iterate basic hill climbing in a loop
  - For constant number of trials
    - Choose random initial state (or use random tie breaking)
    - Run Hill Climbing
    - Record Solution
  - Return largest solution over all trials

Local Beam Search

- Another way to fix local optimum problem
- Since HC only maintains the current state, if it gets stuck in a state, it can't recover
- Beam search maintains a "beam" (open list) of k states with the highest function value
- At each step, successors of all k states are generated and the best k of those are selected, and the search repeats

Example: k=2





Note: This is different from running several HCs in parallel

Example: 8-puzzle

- Associate each state with function value, e.g. “sum of moves to home location”
  - Probably better than “number of misplaced tiles”

- Do beam search to minimize this function to zero

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

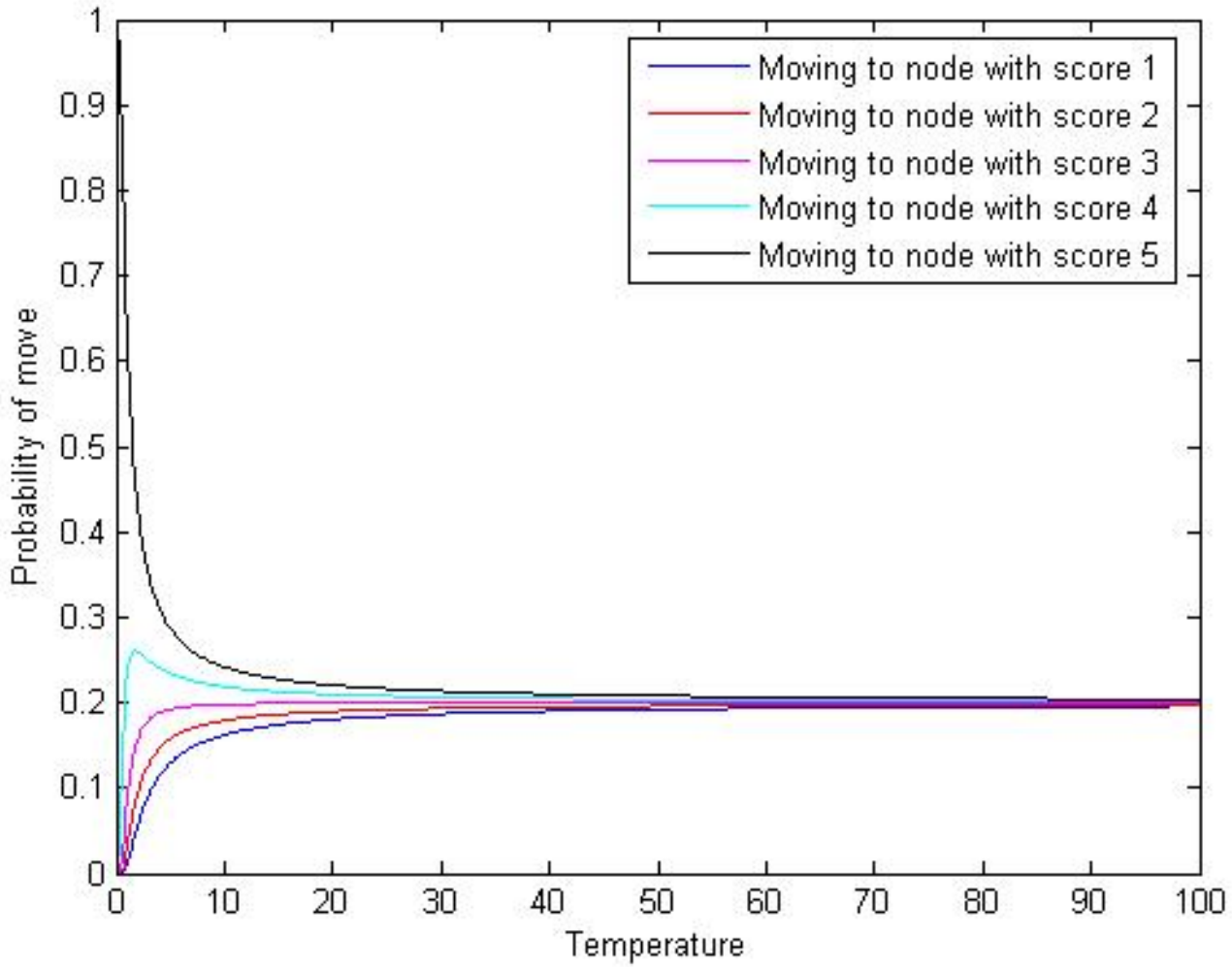
SA Move Selection

- 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"

Example



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