# Lecture 6

# Local Search

**Lusi Li**
**Department of Computer Science**
**ODU**

Reading for This Class:
Chapter 4, Russell and Norvig

# Review

- **Last Class**
  - Informed Search
  - Greedy Best-first Search
  - A* Search
  - Heuristic Functions

- **This Class**
  - Local Search
    - Hill-Climbing
    - Simulated Annealing

- **Next Class**
  - Global Search
    - Genetic Algorithms

# Path Search VS. Local Search

- **The search algorithms we discussed before are designed to find a goal state from a start state s**
  - **the path to the goal that constitutes a solution to the problem**
  - **Uninformed search: g(s)**
  - **Informed search: h(s), g(s)+h(s)**
- **In many optimization problems**
  - **the path to the goal is irrelevant**
  - **the goal state itself is the solution**
- **Another category of search problem**
  - **Local Search Problem**
    - **Never worry about the path**
    - **Just want the goal**
  - **Examples**
    - **Integrated-circuit Design**
    - **Factory-floor layout**
    - **Automatic programming**
    - **Telecommunications network optimization**

# Path Search VS. Local Search

- Path Search maintains a search tree to find the path
  - keep paths in memory and remember alternatives
  - can backtrack

- Local search uses a single search path of solutions, not a search tree
  - start from an initial state
  - at each step consider the current state, and try to improve it by moving only to one of its neighbors (not the one in frontier set)

    ➔ Iterative improvement algorithms
  - No frontier set and no backtracking
  - Each state is a solution

# Local Search Algorithms

- **Objective (Fitness) Function *f(s)***
  - All local search problems have an objective function to specify how "good" a state is
  - Each state *s* has a score *f(s)*
  - The goal is to find the state with the highest (or the lowest) score, or a reasonably high (or low) score
- **General Procedure**
  - Keep only a single (complete) state in memory
  - Generate only the neighbours of that state
  - Keep one of the neighbors and discard others
- **Two strategies for choosing the state to visit next**
  - Hill climbing
  - Simulated annealing
- **Then, an extension to multiple current states**
  - Genetic algorithms

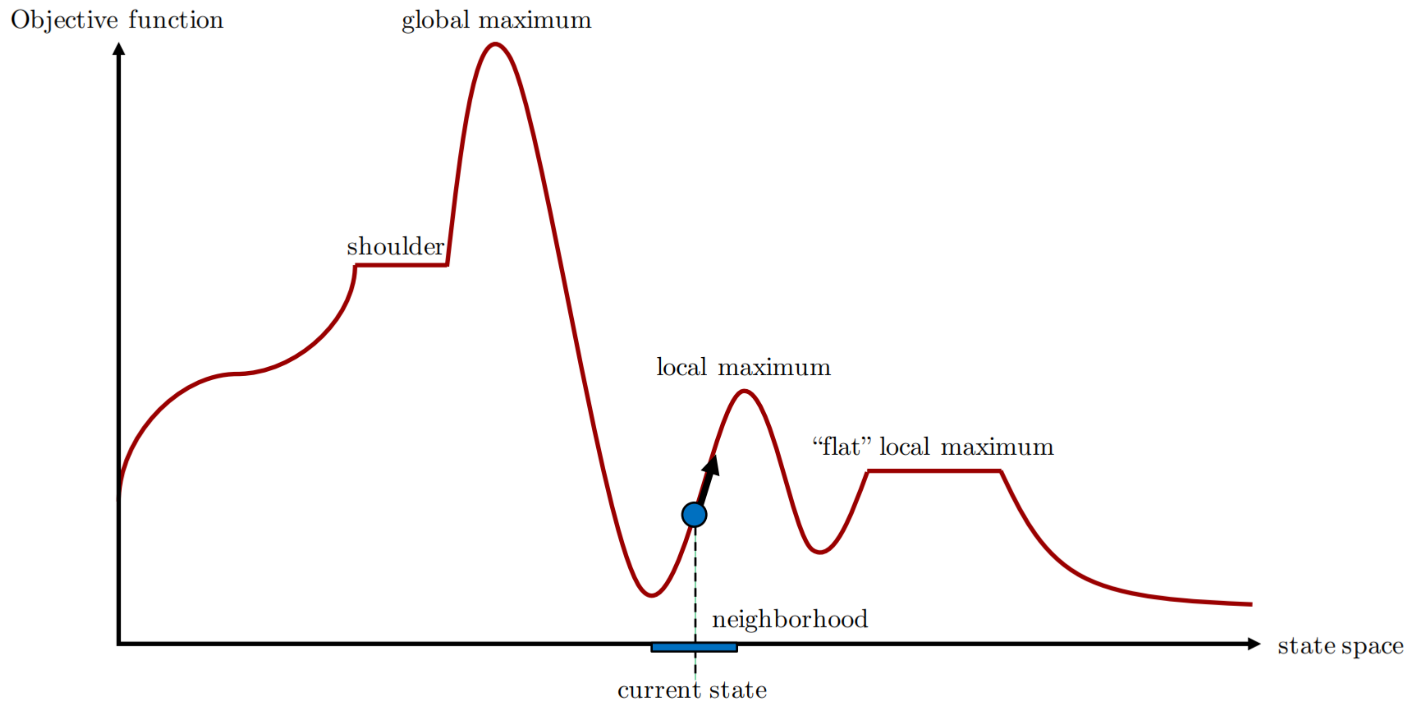
# Local Search Algorithms

- **Two key advantages**
  - Very little memory required
  - Often find reasonable solutions in large or infinite state spaces

- **Usage**
  - Pure optimization problem
  - Find or approximate the best state according to some objective function
  - Optimal if the space to be searched is convex

# 1-D State Space Landscape

- **Global maximum**
  - **Find the highest peak**
- **Global minimum**
  - **Find the lowest valley**




A one-dimensional state-space landscape in which elevation corresponds to the objective function.

# Hill Climbing Algorithm

**function** HILL-CLIMBING(*problem*) **returns** a state that is a local maximum

$current \leftarrow$ MAKE-NODE(*problem*.INITIAL-STATE)
**loop do**
  $neighbor \leftarrow$ a highest-valued successor of $current$
  **if** neighbor.VALUE $\boxed{\leq}$ current.VALUE **then return** *current*.STATE
  $current \leftarrow neighbor$

- **Idea: start from some state s**
  - **move to a neighbor t with a better score f(t). Repeat.**
- **Properties:**
  - **Terminate when no neighbor has better value**
  - **Does not look ahead beyond the immediate neighbors of the current state**
  - **Choose randomly among the set of best successors, if there is more than one**
  - **Do not backtrack, since it doesn't remember where it's been**
  - **Required data structure: the current state and the f(s)**
  - **a.k.a. greedy local search**

# Hill Climbing Algorithm

- **Q1: What’s a neighbor?**
- **Q2: Pick which neighbor?**
- **Q3: What if no neighbor is better than the current state?**

# Hill Climbing Algorithm

- **Q1: What’s a neighbor?**
  - You have to define that!
  - The neighborhood of a state is the set of neighbors.
  - Also called ‘move set’
  - Similar to successor function

- **Q2: Pick which neighbor?**
  - The best one (greedy) based on objective function values

- **Q3: What if no neighbor is better than the current state?**
  - Stop

# Hill-climbing: 8-Queens problem

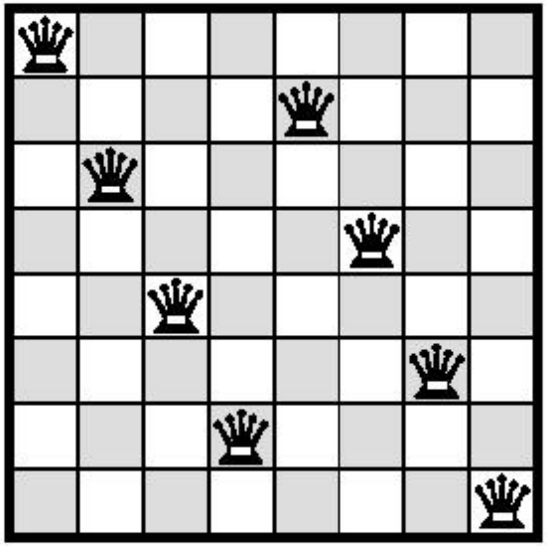
- **Put all 8 queens on the 8 x 8 board with no two queens attacking each other, i.e, no two queens can share the same row, column, or diagonal.**
- **Complete state formulation:**
  - **State:**
  - **Neighbor states:**
  - **Fitness function f:**

**Constraints:**

1.Each row must contain exactly one queen.

2.Each column must contain exactly one queen.

3.No two queens should be in the same row, column, or diagonal.



# Hill-climbing: 8-Queens problem

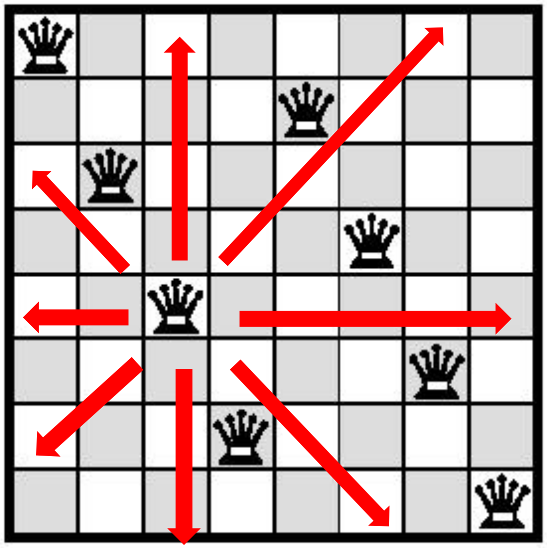
- **Put all 8 queens on the 8 x 8 board with no two queens on the same row, column, or diagonal**
- **Complete state formulation:**
  - **State: positions of the 8 Queens one per column**
  - **Neighbor states: generated by moving one queen to a different square in the same column**
  - **Fitness function f: number of pairs of queens that are attacking each other**
    - **Note that we want a state s with the lowest score f(s) = 0**
    - **Low or high should be obvious from context.**

**Constraints:**

1.Each row must contain exactly one queen.

2.Each column must contain exactly one queen.

3.No two queens should be in the same row, column, or diagonal.



# 8-Queens problem: fitness values of neighborhood

**f(s) = 3 + 4 + 2 + 3 + 2 + 2 + 1 + 0 = 17**

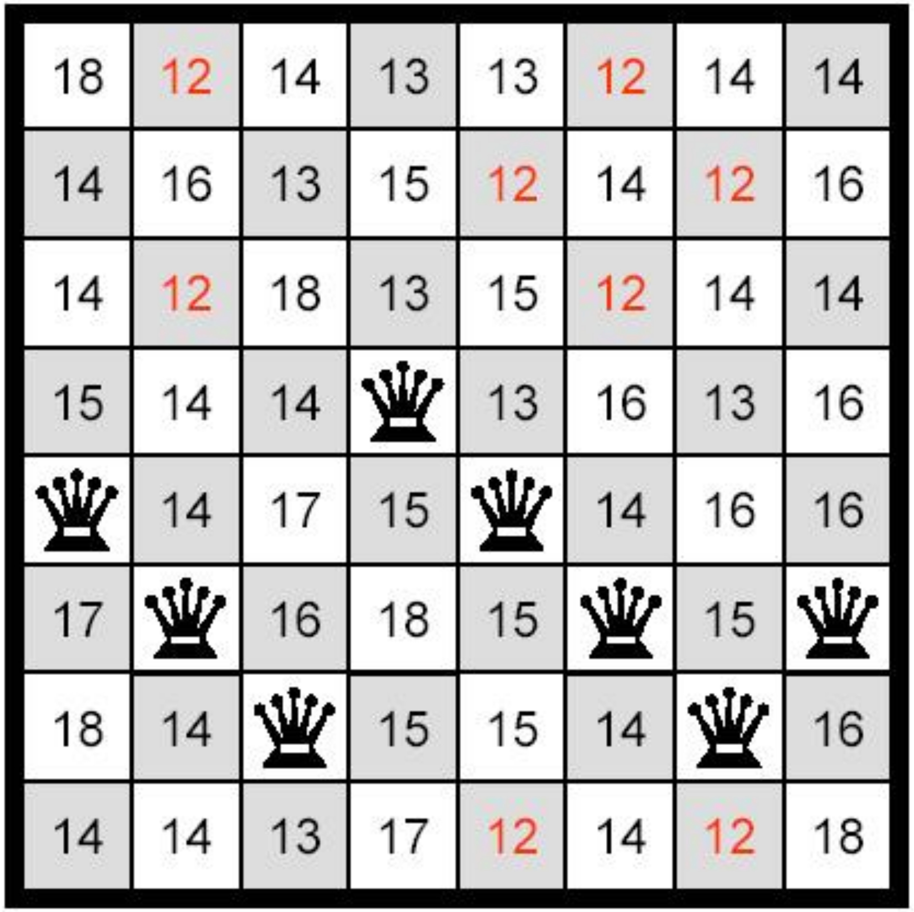



**Fig. 1** An 8-queens state **s**

An 8-queens state with f(s) = 17.

It also shows the value of f for each possible successor obtained by moving a queen within its column, with the best one having f = 12.

The best moves are marked.

**Hill-climbing algorithms typically choose randomly among the set of best successors if there is more than one.**

# 8-Queens problem: Local minimum

5 steps

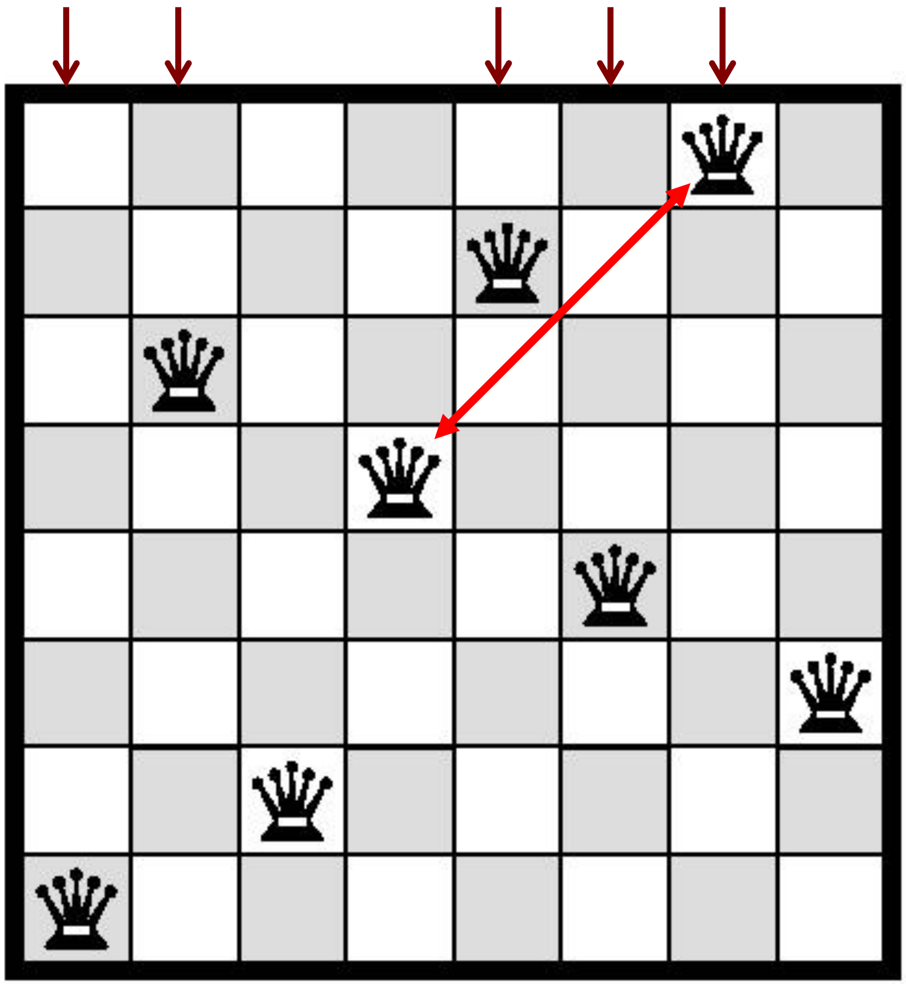


Fig. 2 An 8-queens state **t**

An 8-queens state with f(t) = 1.

It is a local optimum because every move leads to a larger f.

# Performance of 8-Queens Problem

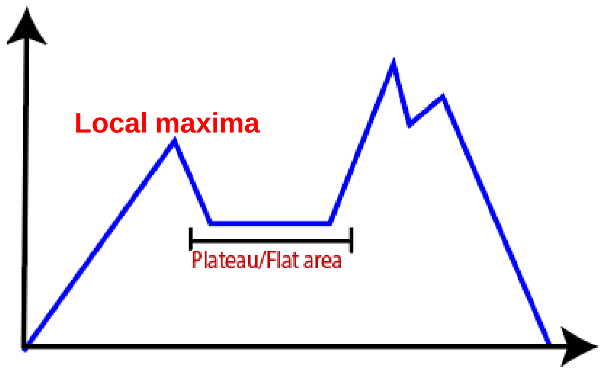
- **8-queens statistics:**
  - **State space of size ≈ 17 million**
    - **Starting from random state, steepest-ascent hill climbing solves 14% of problem instances and gets stuck for 86% of problem instances**
    - **It takes 4 steps on average when it succeeds, 3 when it gets stuck**
  - **When sideways moves are allowed, performance improve**
    - **Sideways moves: if no uphill moves, allow moving to a state with the same value as the current one.**
    - **E.g., 100 consecutive sideways moves, 14% -> 94%**

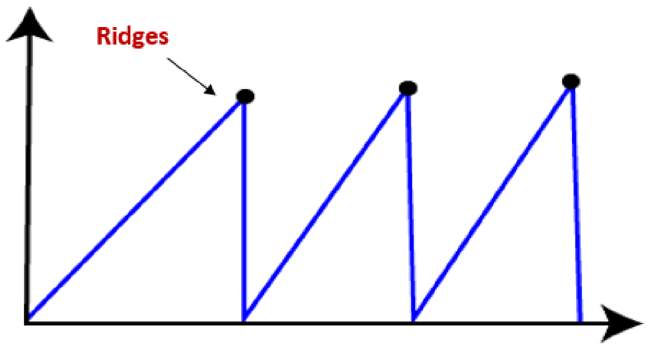
# Analysis of Hill-Climbing

- **Continually moves uphill**
  - **increasing value of the evaluation function**
    - **(or “downhill” decreasing value of the cost function)**
  - **gradient descent search is a variation that moves downhill**
- **Very simple strategy with low space requirements**
  - **stores only the state and its evaluation, no search tree**
- **Problems**
  - **local maxima**
  - **plateau**
  - **ridges**

# Analysis of Hill-Climbing

- **Problems**
  - local maxima
    - the peak is higher than all its neighbors, but not the global maximum
    - algorithm can't go higher, but is not at a satisfactory solution
  - plateau
    - area where the evaluation function is flat
    - the best neighborhood has the same value as the current state
  - ridges
    - sequence of local maxima difficult for greedy algorithms to navigate
    - search may oscillate slowly



# Further Variants of Hill Climbing

- **Sideways moves:**
  - **if no uphill moves, allow moving to a state with the same value as the current one (escape shoulders)**
- **Stochastic hill-climbing:**
  - **selection among the available uphill moves is done to be "less" greedy**
  - **the better, the more likely**
- **First-choice hill-climbing:**
  - **successors are generated randomly, one at a time, until one that is better than the current state is found**
  - **if better, take the move**
  - **deal with large neighborhoods**
- **Random-restart hill climbing:**
  - **conducts a series of hill-climbing searches from randomly generated initial states, until a goal is found.**
  - "If at first you don't succeed, try, try again."

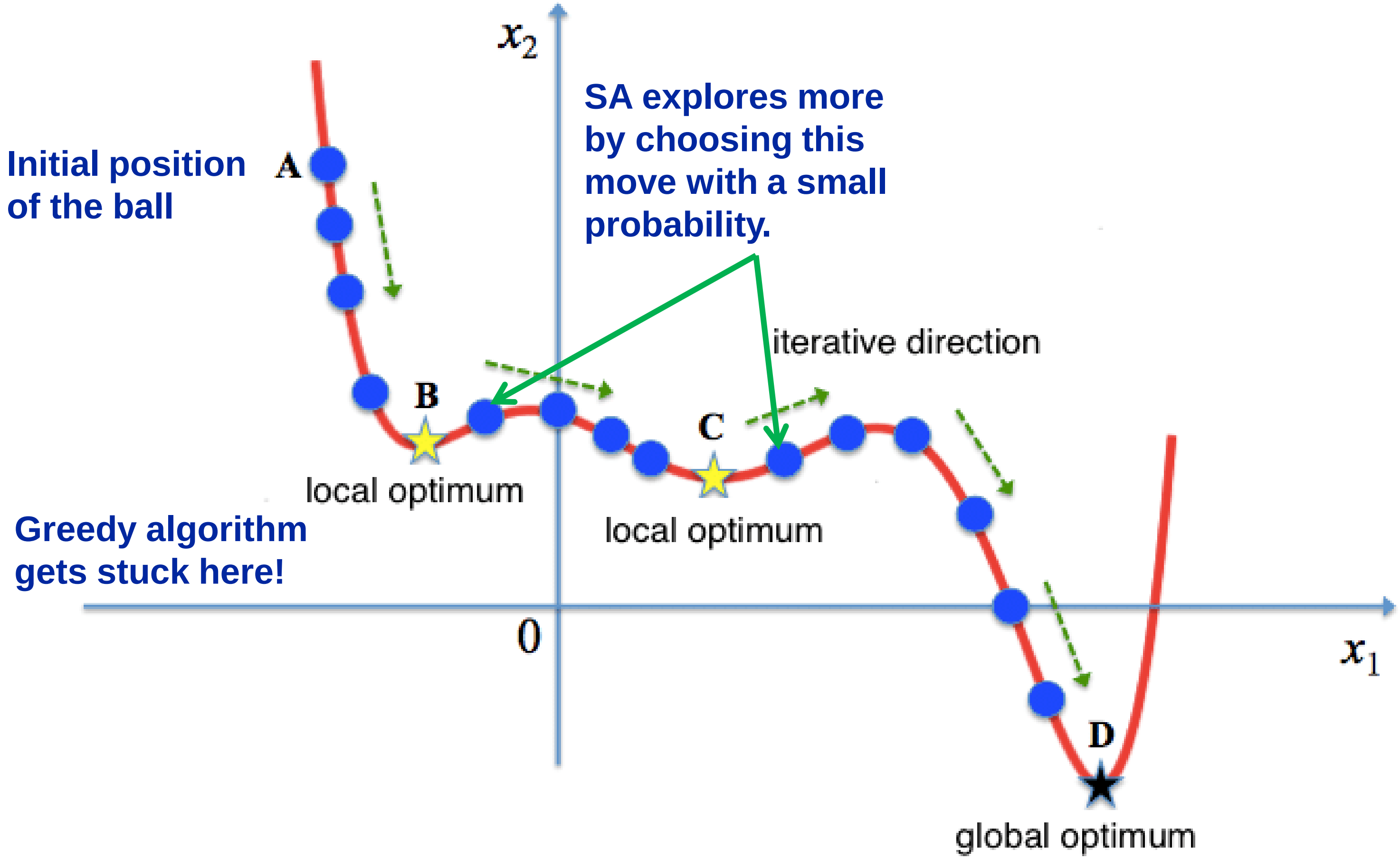
# Simulated Annealing

- **Escape from local optima**
  - by accepting, with a probability that decreases during the search, also moves that are worse than the current solution (going "downhill")
- **Inspired by the process of annealing of solids in metallurgy:**
  - annealing: harden metals and glass by heating them to a high temperature and then gradually cooling them, thus allowing the material to reach a low-energy crystalline state
  - at the start, make lots of moves and then gradually slow down

# Simulated Annealing: Intuition

- **Minimization problem**
- **Imagine a state space landscape on table**
- **Let ping-pong ball from random point → local minimum**
- **Shake table →ball tends to find different minimum**
- **Shake hard at first (high temperature) but gradually reduce intensity (lower temperature)**

# Ball on terrain example – SA vs Greedy Algorithms




Upon a large no. of iterations, SA converges to the solution.

# Simulated Annealing

1. Pick an initial state s
2. Randomly pick t in neighbors(s)
3. IF f(t) better THEN accept s ← t.
4. ELSE /* t is worse than s */
5. accept s ← t **with a small probability**
6. GOTO 2 until bored.

**Q: How to choose the small probability?**

# Simulated Annealing

1. Pick an initial state s
2. Randomly pick t in neighbors(s)
3. IF f(t) better THEN accept s ← t.
4. ELSE /* t is worse than s */
5. accept s ← t with a small probability
6. GOTO 2 until bored.

Q: How to choose the small probability?

- idea 1: p = 0.1
- idea 2: p decreases with time
- idea 3: p decreases with time, also as the difference between f(t) and f(s) increases

# Simulated Annealing

1. Pick initial state s
2. Randomly pick t in neighbors(s)
3. IF f(t) better THEN accept s ← t.
4. ELSE /* t is worse than s */
5. accept s ← t with a small probability
6. GOTO 2 until bored.

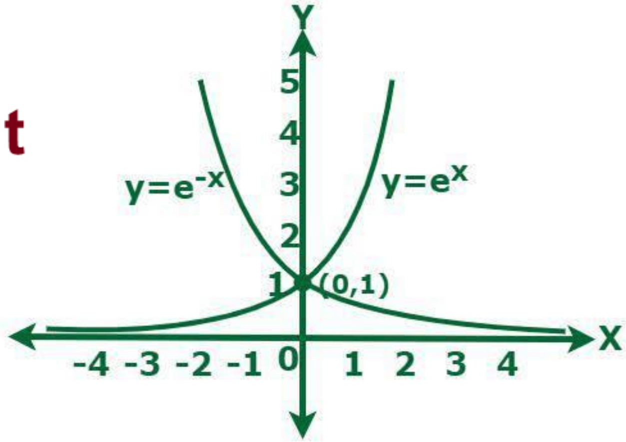
**Q: How to choose the small probability?**

- idea 1: p = 0.1
- idea 2: p decreases with time
- idea 3: p decreases with time, also as the difference between f(t) and f(s) increases

# Simulated Annealing

- $\Delta E = f(t) - f(s)$
- If f(t) is better than f(s), always accept t
- Otherwise, accept t with probability

$$p = e^{\frac{\Delta E}{T}}$$




  - i.e., if $r < p$ ($r \in [0, 1]$ is a uniform random number), accept t
  - where T is a temperature parameter that 'cools' (anneals) over time, e.g. T $\leftarrow$ T * 0.9
    - High T allows more worse moves
    - Low T results in few or no bad moves

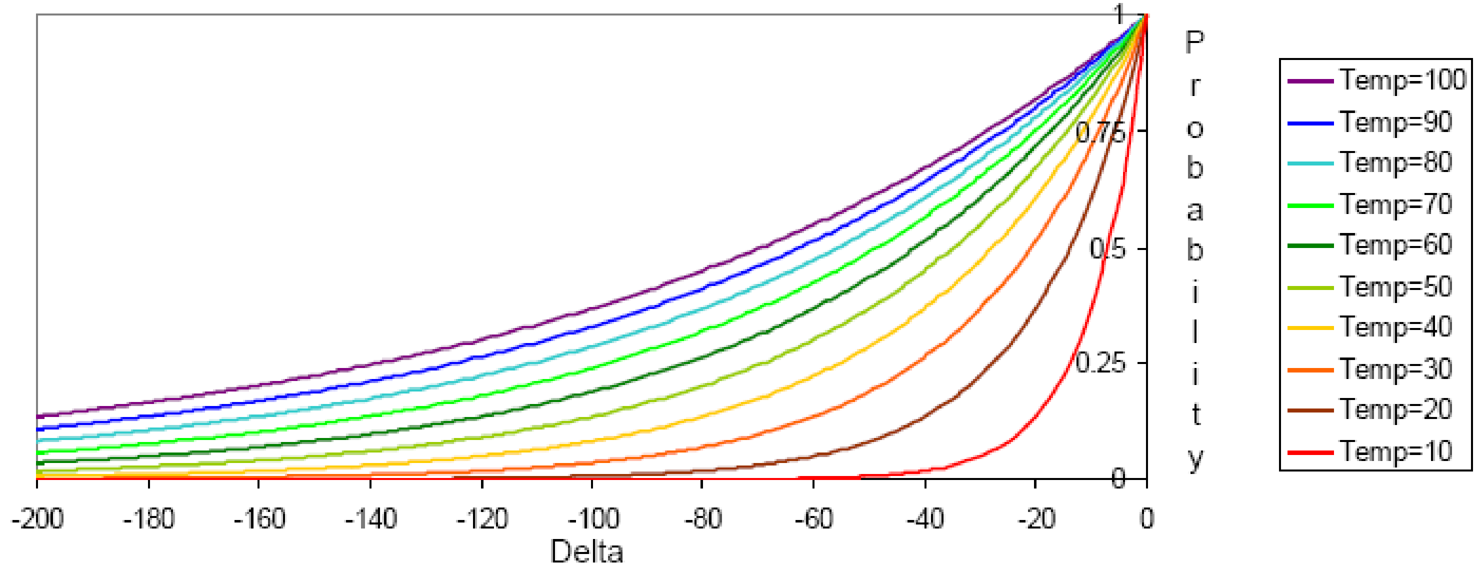
- If the difference (formally known as energy difference) |f(t)-f(s)| is large, the probability is small.

# Simulated Annealing

- **Acceptance criterion and cooling schedule**

if (delta>=0) accept

else if ($random < e^{delta/Temp}$) accept, else reject /* 0<=random<=1 */




Initially temperature is very high (most bad moves accepted)

Temp slowly goes to 0, with multiple moves attempted at each temperature

Final runs with temp=0 (always reject bad moves) greedily “quench” the system

# Simulated Annealing

**function** SIMULATED-ANNEALING($problem, schedule$) **returns** a solution state
  **inputs**: $problem$, a problem
    $schedule$, a mapping from time to "temperature"

  $current \leftarrow$ MAKE-NODE($problem$.INITIAL-STATE)
  **for** $t$ = 1 **to** $\infty$ **do**
    $T \leftarrow schedule(t)$ **// T is the current temperature, which is monotonically decreasing with t**
    **if** $T$ = 0 **then return** $current$ **//halt when temperature = 0**
    $next \leftarrow$ a randomly selected successor of $current$
    $\Delta E \leftarrow next$.VALUE – $current$.VALUE **// If positive, next is better than current.**
    **if** $\Delta E > 0$ **then** $current \leftarrow next$ **// Otherwise, next is worse than current.**
    **else** $current \leftarrow next$ only with probability $e^{\Delta E/T}$

**// as T → 0, p → 0; as ΔE → - ∞, p → 0**

# Summary

- **Local Search**
- **Hill-climbing**
- **Simulated Annealing**

# What I want you to do

- **Review Chapter 4**