CS 540 Introduction to Artificial Intelligence
Advanced Search

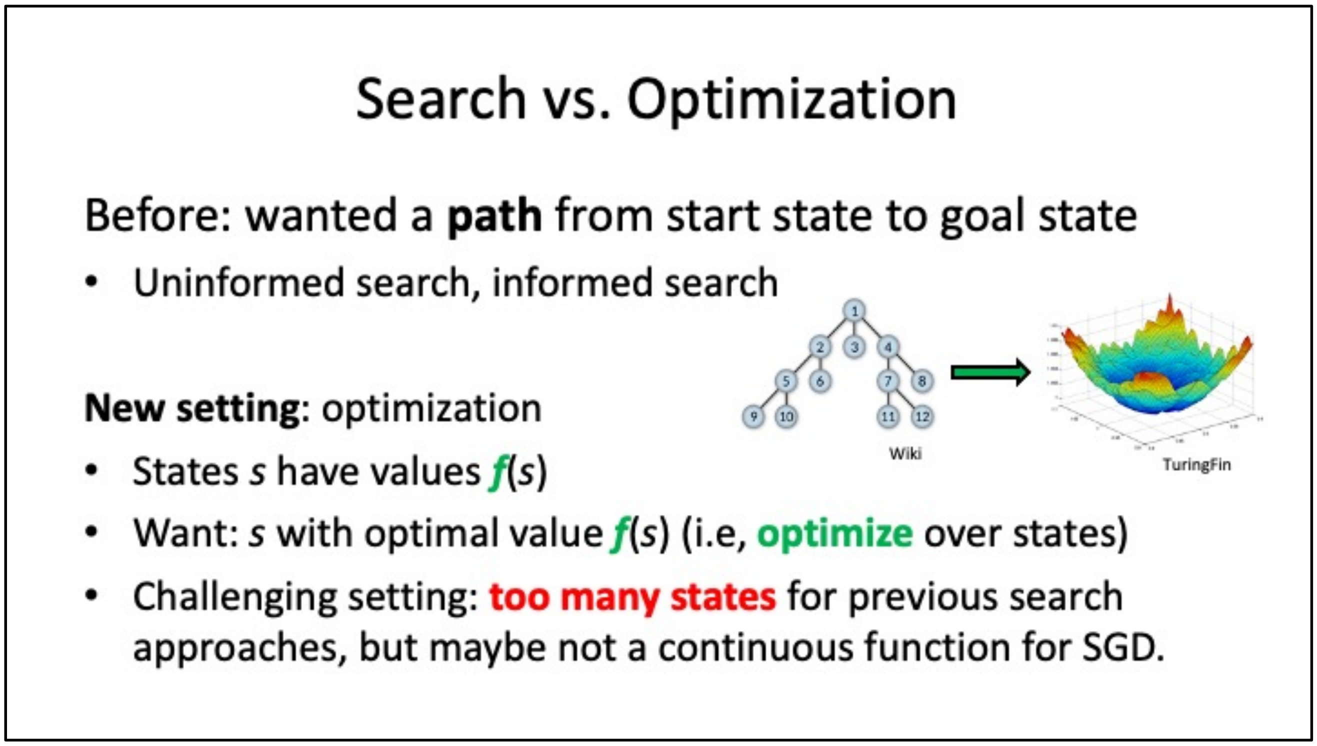
Yingyu Liang
University of Wisconsin-Madison

Nov 18, 2021

Based on slides by Fred Sala

Outline

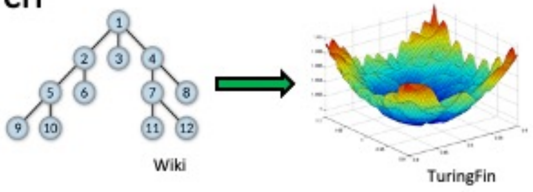
- Advanced Search & Hill-climbing
  - More difficult problems, basics, local optima, variations
- Simulated Annealing
  - Basic algorithm, temperature, tradeoffs
- Genetic Algorithms
  - Basics of evolution, fitness, natural selection



Search vs. Optimization

Before: wanted a **path** from start state to goal state

- Uninformed search, informed search

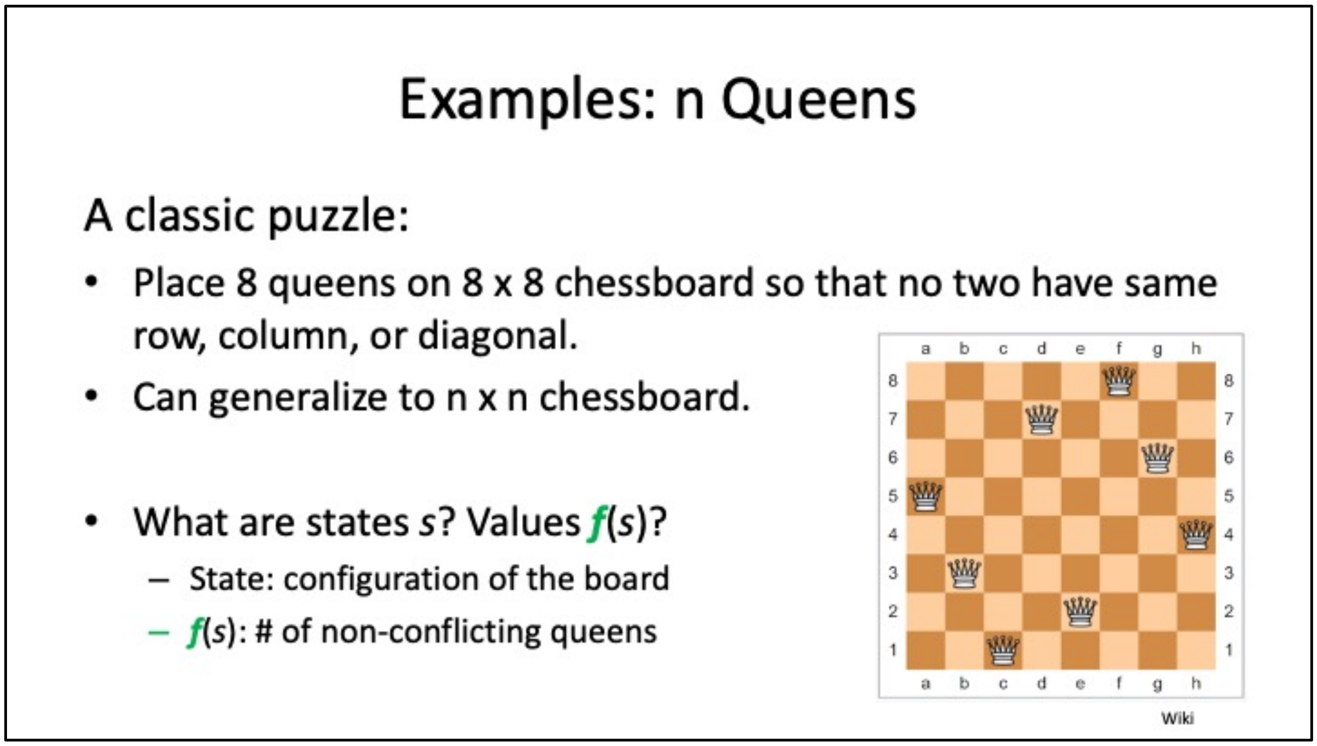


New setting: optimization

- States *s* have values ***f***(*s*)
- Want: *s* with optimal value ***f***(*s*) (i.e, **optimize** over states)
- Challenging setting: **too many states** for previous search approaches, but maybe not a continuous function for SGD.

Advanced search algorithms are designed for a slightly different problem form: optimization.

Key differences:
1. Has values
2. Want the best state, do not care about the path

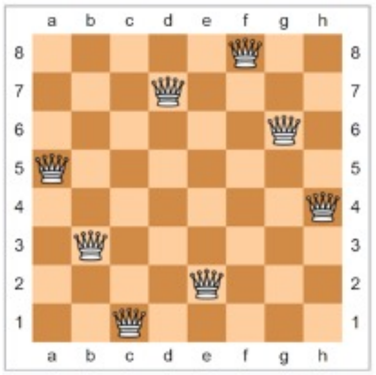


Examples: n Queens

A classic puzzle:

- Place 8 queens on 8 x 8 chessboard so that no two have same row, column, or diagonal.
- Can generalize to n x n chessboard.

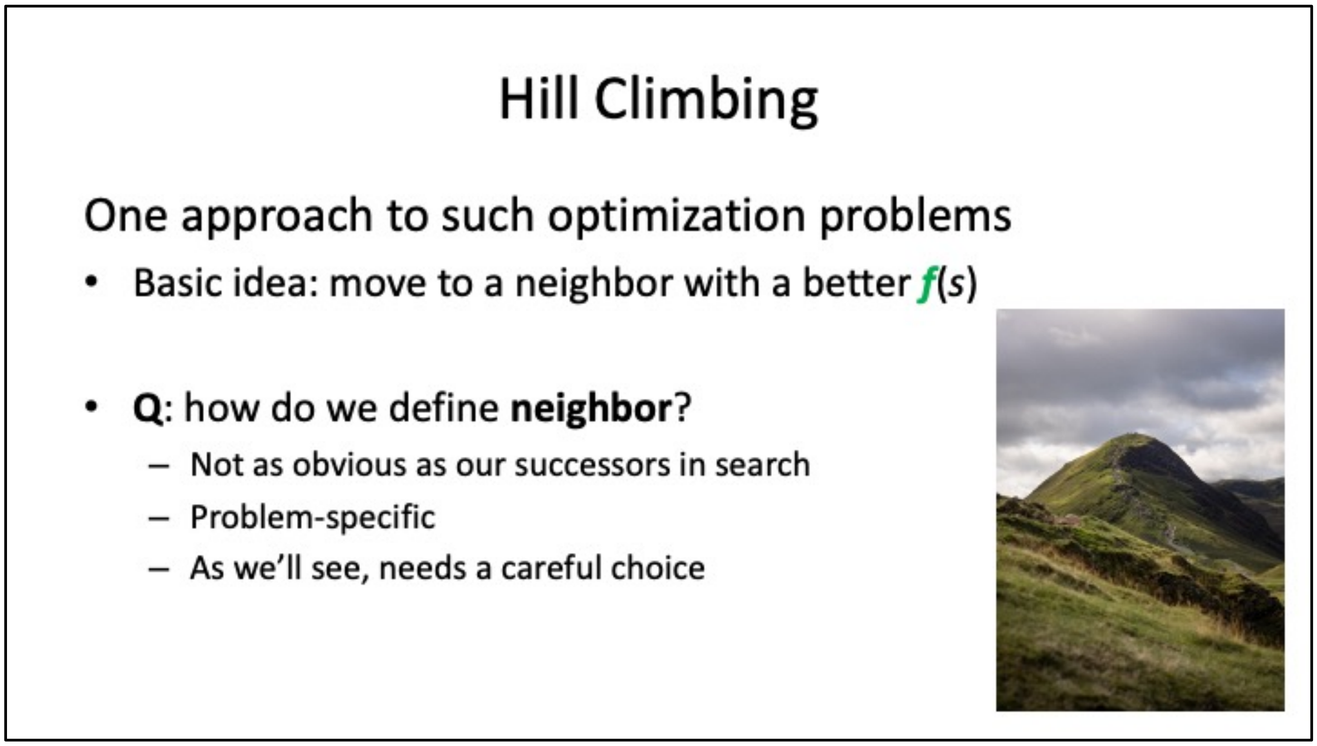
- What are states *s*? Values ***f***(*s*)?
  - State: configuration of the board
  - ***f***(*s*): # of non-conflicting queens



Wiki

State: configuration of the board. Usually we consider a simplification as follows: we only consider those the configurations where each column has one queen.

f(s): can be # of non-conflicting queens, or # of non-conflicting pairs of queens, and we would like to maximize f. It can also be # of conflicting queens, and we would like to minimize f.



Hill Climbing

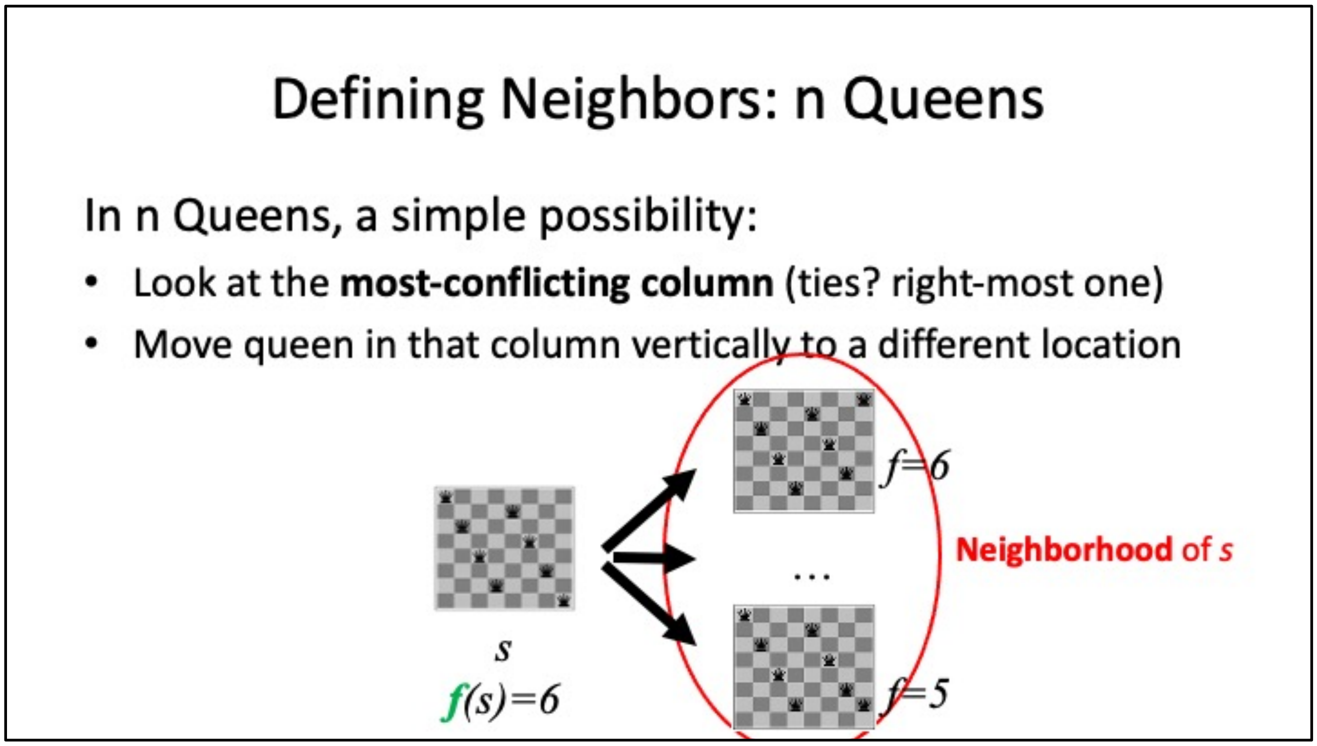
One approach to such optimization problems

- Basic idea: move to a neighbor with a better *f*(*s*)
- **Q**: how do we define **neighbor**?
  - Not as obvious as our successors in search
  - Problem-specific
  - As we'll see, needs a careful choice

Basic idea: iteratively move to a neighbor with a better value. greedy based on local information.

Key question: definition of neighborhood.

1. Determines the performance of the algorithm.
2. Subtle difference from the successor function. Successors are typically defined by constraints in the search problem. Neighbors are more of a design choice.
3. The definition is usually problem specific, and needs a careful choice.



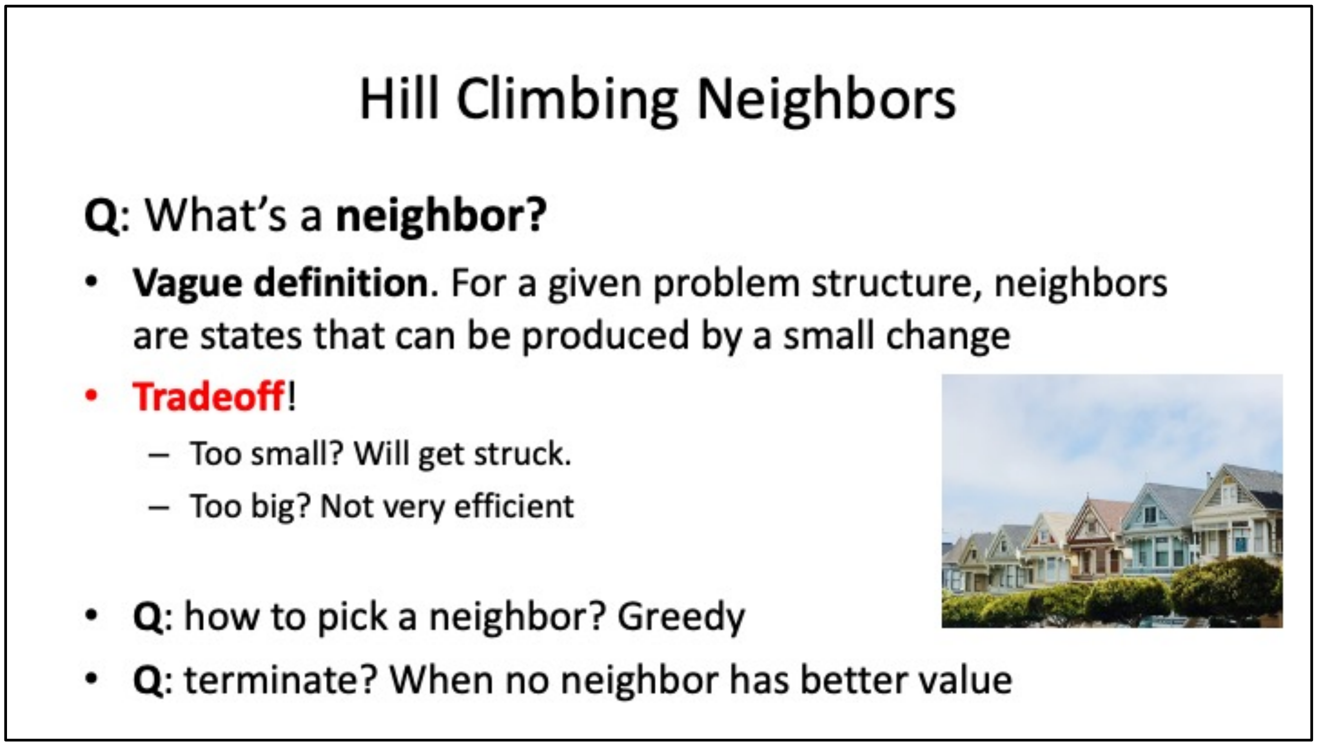
Defining Neighbors: n Queens

In n Queens, a simple possibility:

- Look at the **most-conflicting column** (ties? right-most one)
- Move queen in that column vertically to a different location

s
f(s)=6
f=6
...
f=5
Neighborhood of s

A concrete example of n queens:
First check each column, find how many conflicts the queen in the column has.
Then pick the most-conflicting column.
Define the neigbhors: by moving the queen in that column to different locations in the same column.



Hill Climbing Neighbors

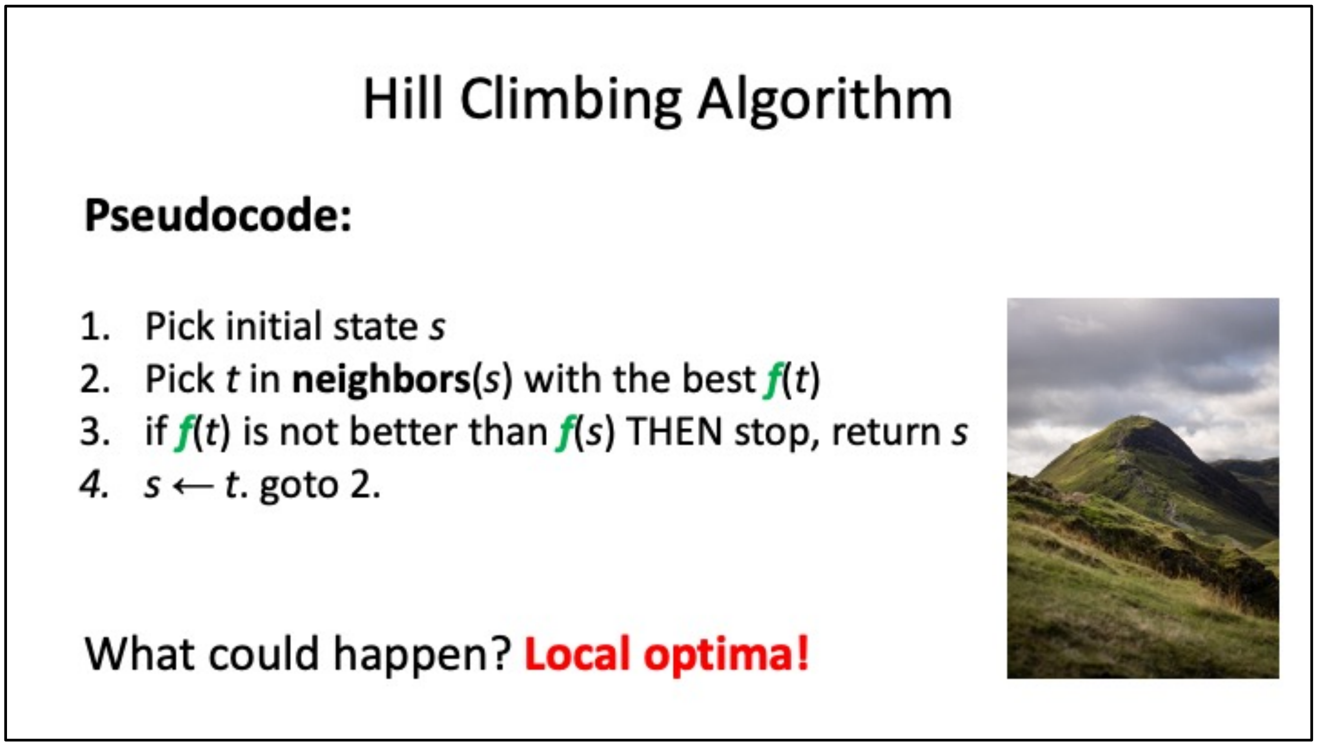
Q: What's a **neighbor?**

- **Vague definition**. For a given problem structure, neighbors are states that can be produced by a small change
- **Tradeoff**!
  - Too small? Will get struck.
  - Too big? Not very efficient

- **Q**: how to pick a neighbor? Greedy
- **Q**: terminate? When no neighbor has better value

In general, neighbors are states that can be produced by a small change. The key is "small", and the change can be problem specific. But how small that should be?

Once we have the definition of neighbors, we can begin to formalize the algorithm:
1. Greedy to pick a neighbor
2. Stopping criteria: no improvement in the neighborhood.



Hill Climbing Algorithm

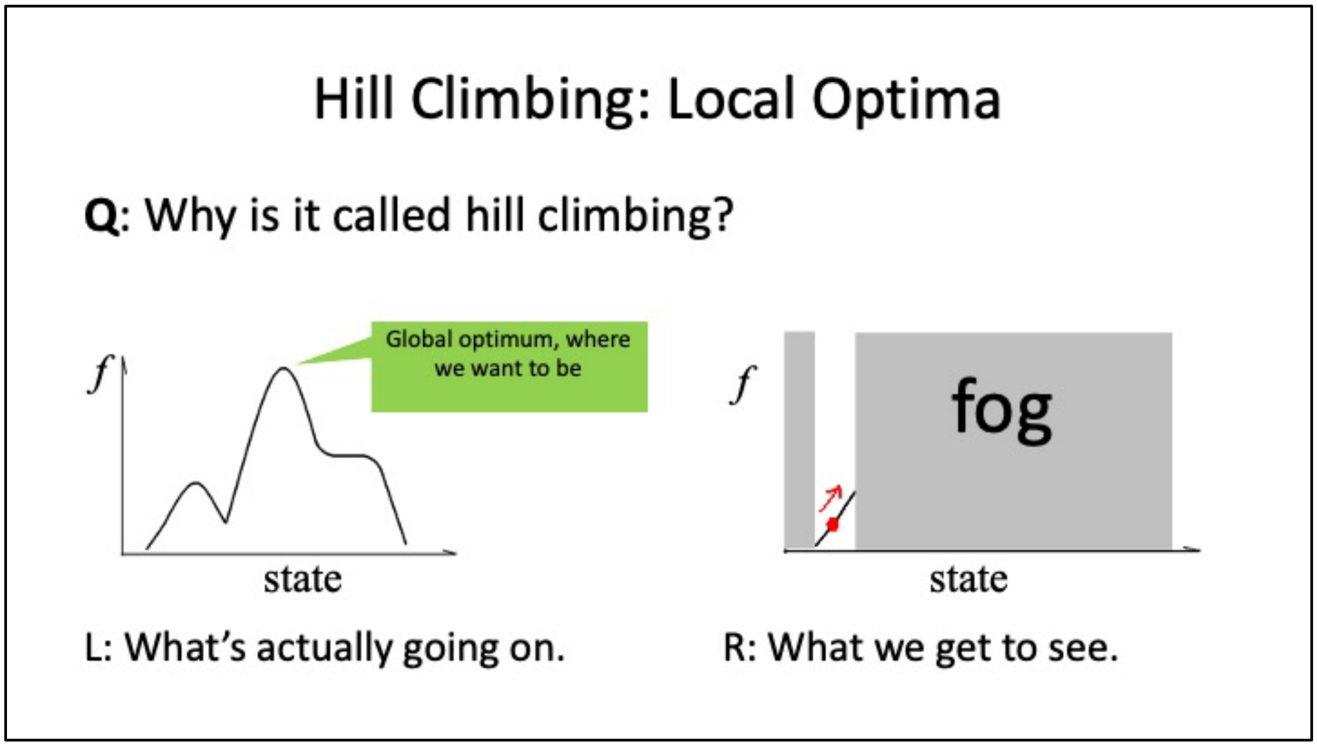
Pseudocode:

```
1. Pick initial state $s$
2. Pick $t$ in neighbors($s$) with the best $f(t)$
3. if $f(t)$ is not better than $f(s)$ THEN stop, return $s$
4. $s \leftarrow t$. goto 2.
```

What could happen? **Local optima!**

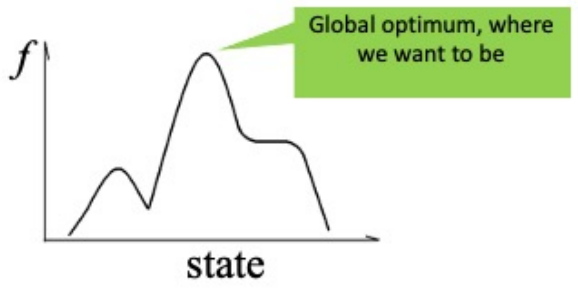
Here is the pseudocode for the basic version of the algo.

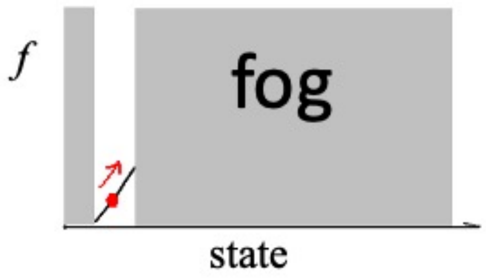
Of course, it may not be able to solve all issues. In particular, it has the issue of local optima.



Hill Climbing: Local Optima

Q: Why is it called hill climbing?





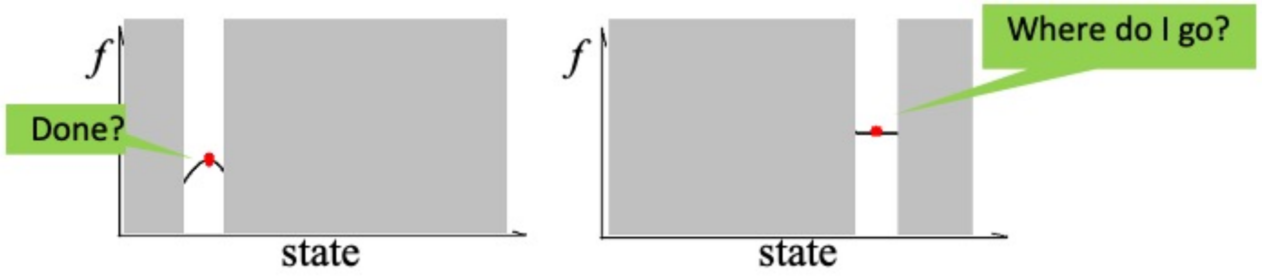
L: What's actually going on.

R: What we get to see.

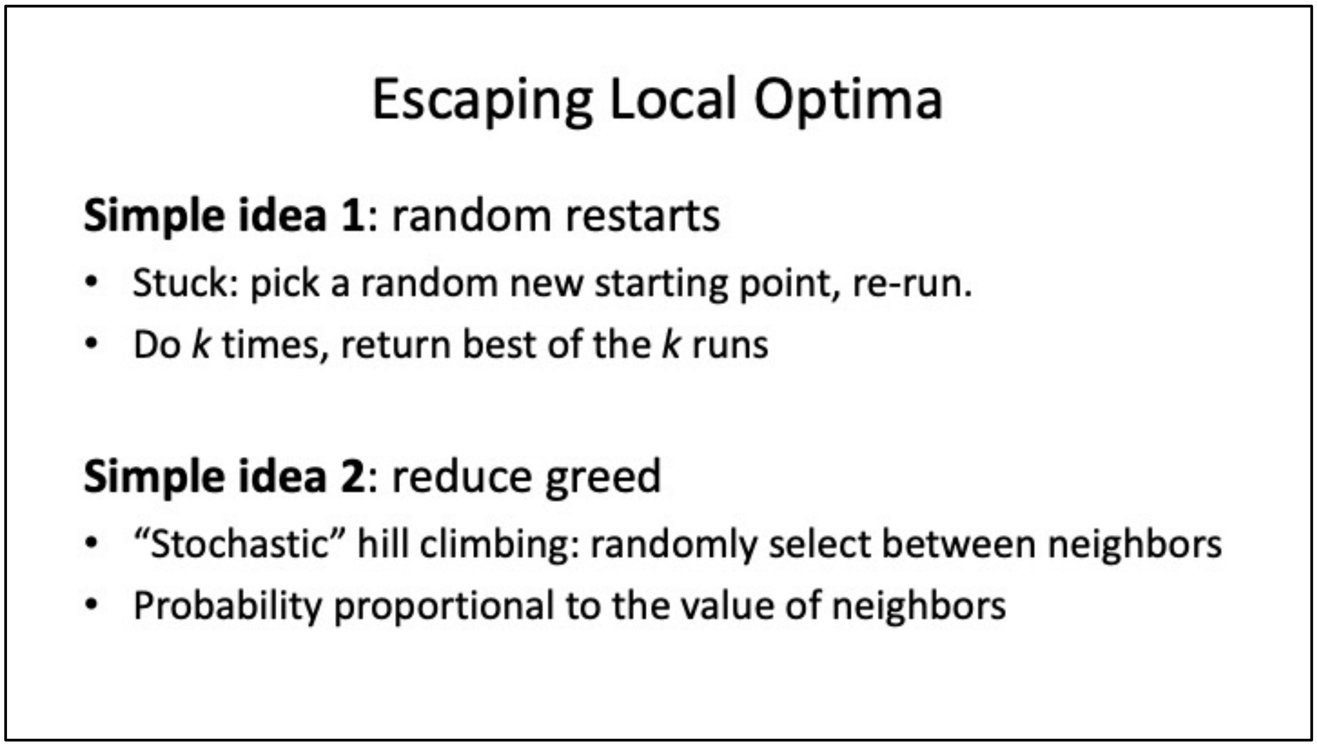
Left: the global picture. We would like to maximize f(x), which is like climbing the hill. Right: but the algorithm can only take action based on information in the neighborhood. It doesn't know the global picture.

Hill Climbing: Local Optima

Note the **local optima**. How do we handle them?



Two challenging cases. The basic algo will just stop and return the current solution.



Escaping Local Optima

Simple idea 1: random restarts

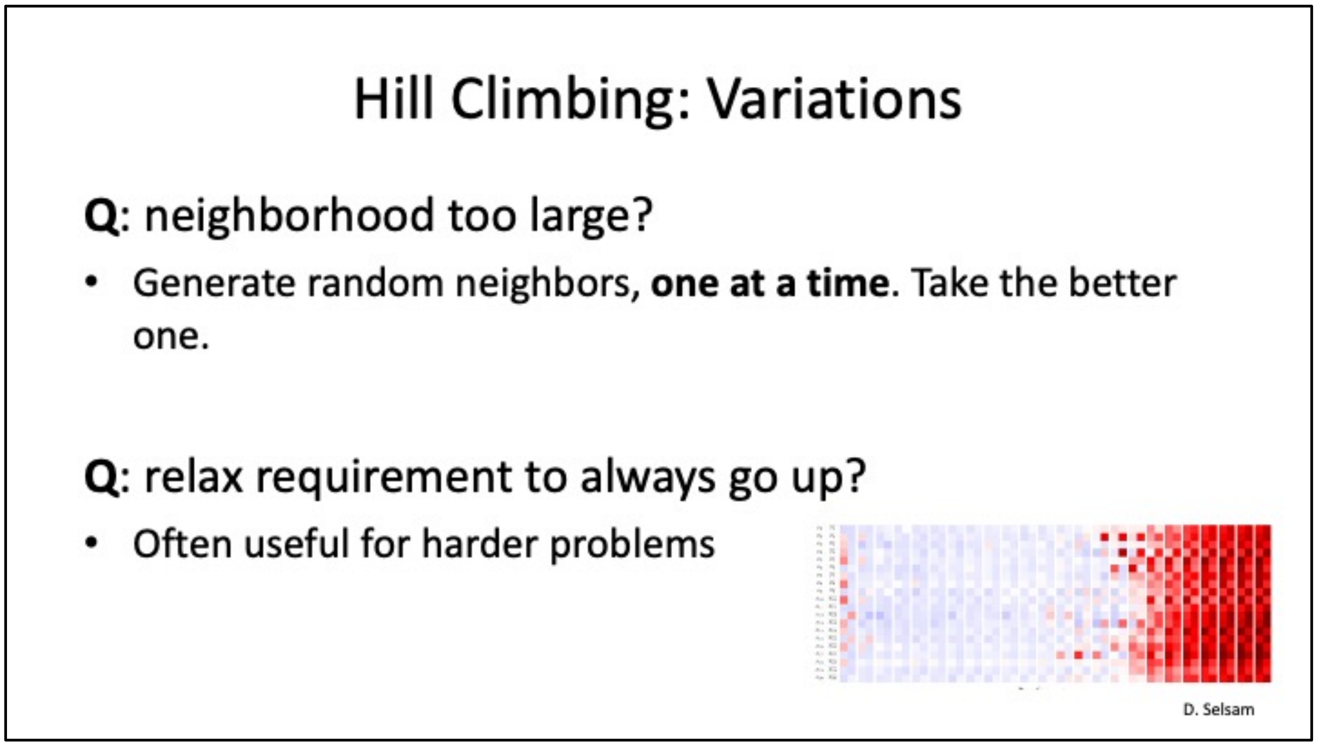
- Stuck: pick a random new starting point, re-run.
- Do *k* times, return best of the *k* runs

Simple idea 2: reduce greed

- "Stochastic" hill climbing: randomly select between neighbors
- Probability proportional to the value of neighbors

Improvement of the basic algo to try to escape local optima. (May not always be able to escape)

1. Random restarts: very simple, and used often in practice.
2. Less greedy: allow to select not-best neighbors, e.g., can be stochastic, picking better neighbors with higher probabilities but still having some probabilities for picking the no-as-good neighbors.



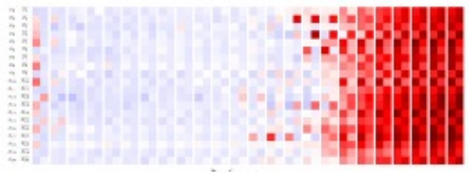
Hill Climbing: Variations

Q: neighborhood too large?

- Generate random neighbors, **one at a time**. Take the better one.

Q: relax requirement to always go up?

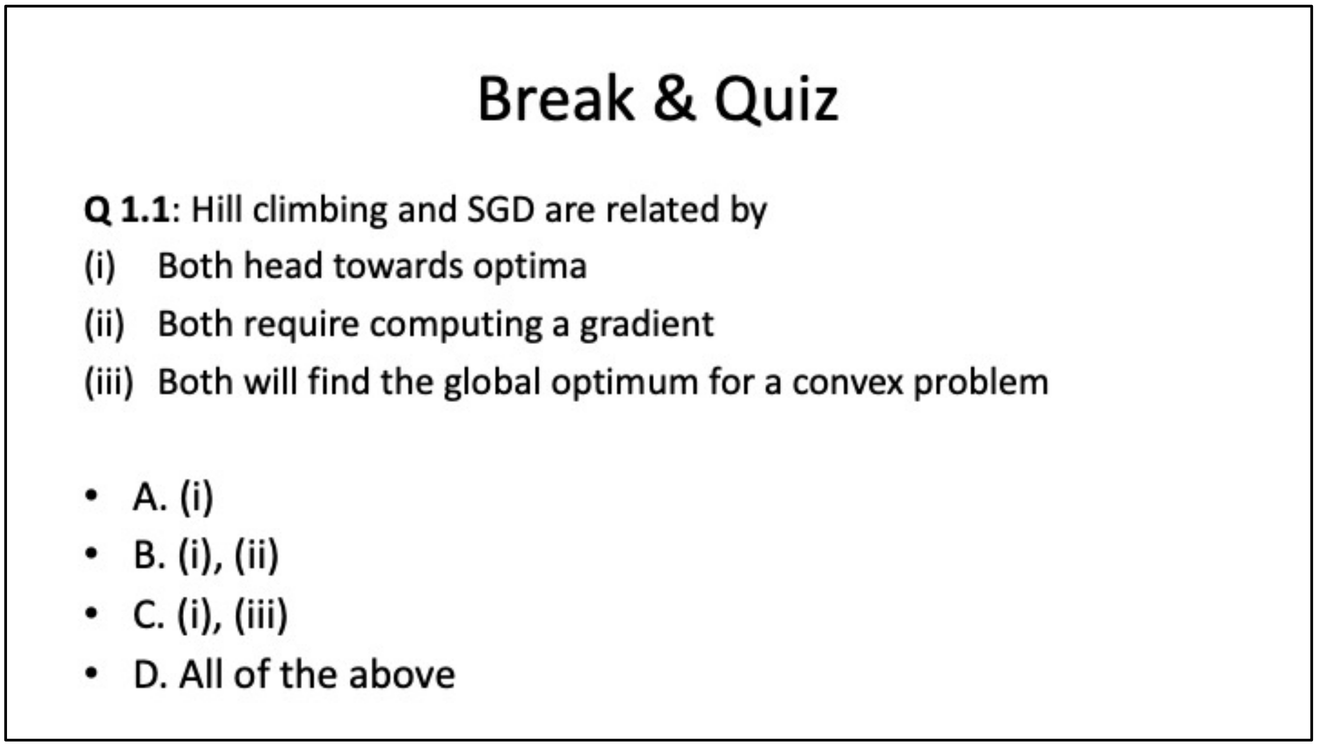
- Often useful for harder problems



D. Selsam

Some more technical comments about the implementation.

1. May not examine the whole neighborhood. Can generate random neighbors one by one and take it once we get a neighbor better than the current state. Or can generate a fixed number of random neighbors, find the best neighbor and compare to the current state.
2. Be less greedy. Like stochastic neighbors as in the previous slide



Break & Quiz

Q 1.1: Hill climbing and SGD are related by

(i) Both head towards optima

(ii) Both require computing a gradient

(iii) Both will find the global optimum for a convex problem

- A. (i)
- B. (i), (ii)
- C. (i), (iii)
- D. All of the above

Break & Quiz

Q 1.1: Hill climbing and SGD are related by

(i) Both head towards optima

(ii) Both require computing a gradient

(iii) Both will find the global optimum for a convex problem

- A. (i)
- B. (i), (ii)
- **C. (i), (iii)**
- D. All of the above

Break & Quiz

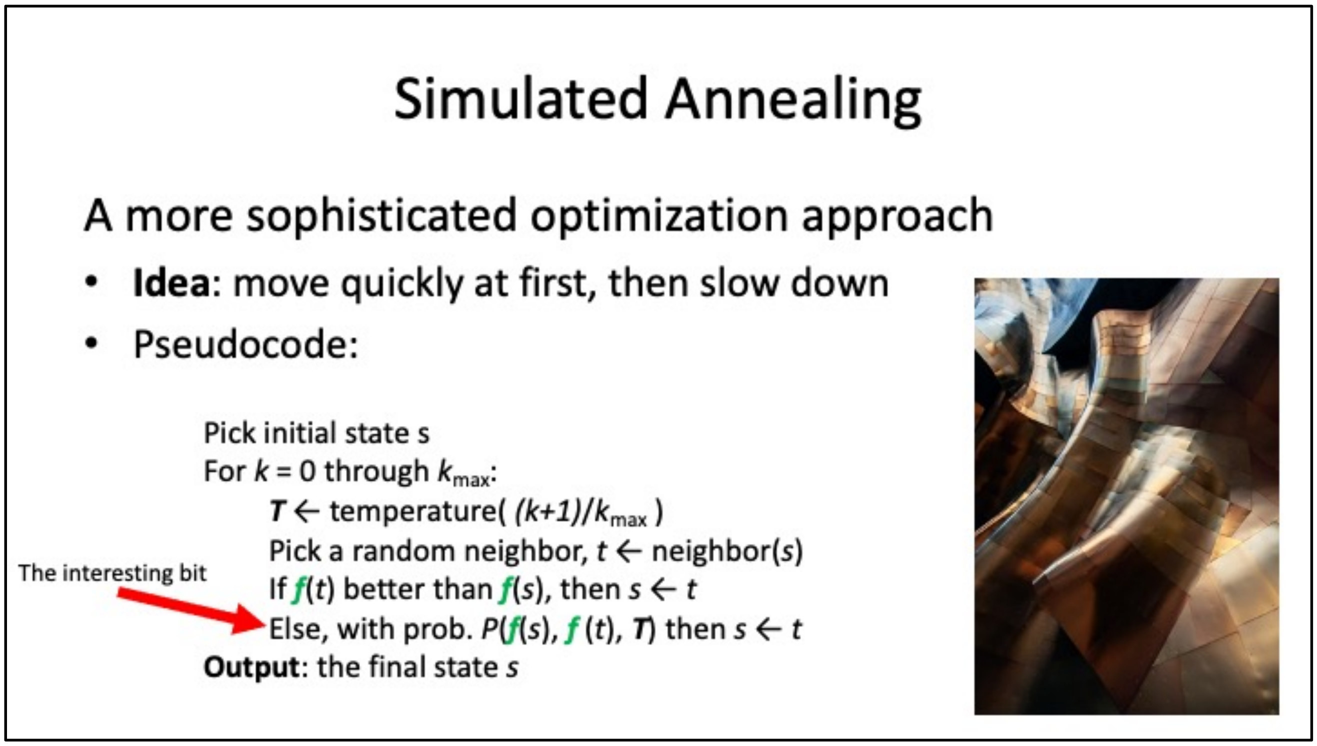
Q 1.1: Hill climbing and SGD are related by

(i) Both head towards optima

(ii) Both require computing a gradient

(iii) Both will find the global optimum for a convex problem

- A. (i) (No: (iii) also true since convexity->local optima are global)
- B. (i), (ii) (No: (ii) is false. Hill-climbing looks at neighbors only.)
- **C. (i), (iii)**
- D. All of the above (No: (ii) false, as above.)



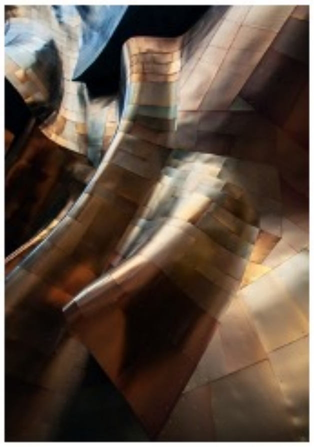
Simulated Annealing

A more sophisticated optimization approach

- **Idea**: move quickly at first, then slow down
- Pseudocode:

```
Pick initial state s
For $k$ = 0 through $k_{max}$:
    $T \leftarrow$ temperature( $(k+1)/k_{max}$ )
    Pick a random neighbor, $t \leftarrow$ neighbor($s$)
    If $f(t)$ better than $f(s)$, then $s \leftarrow t$
    Else, with prob. $P(f(s), f(t), T)$ then $s \leftarrow t$
Output: the final state $s$
```

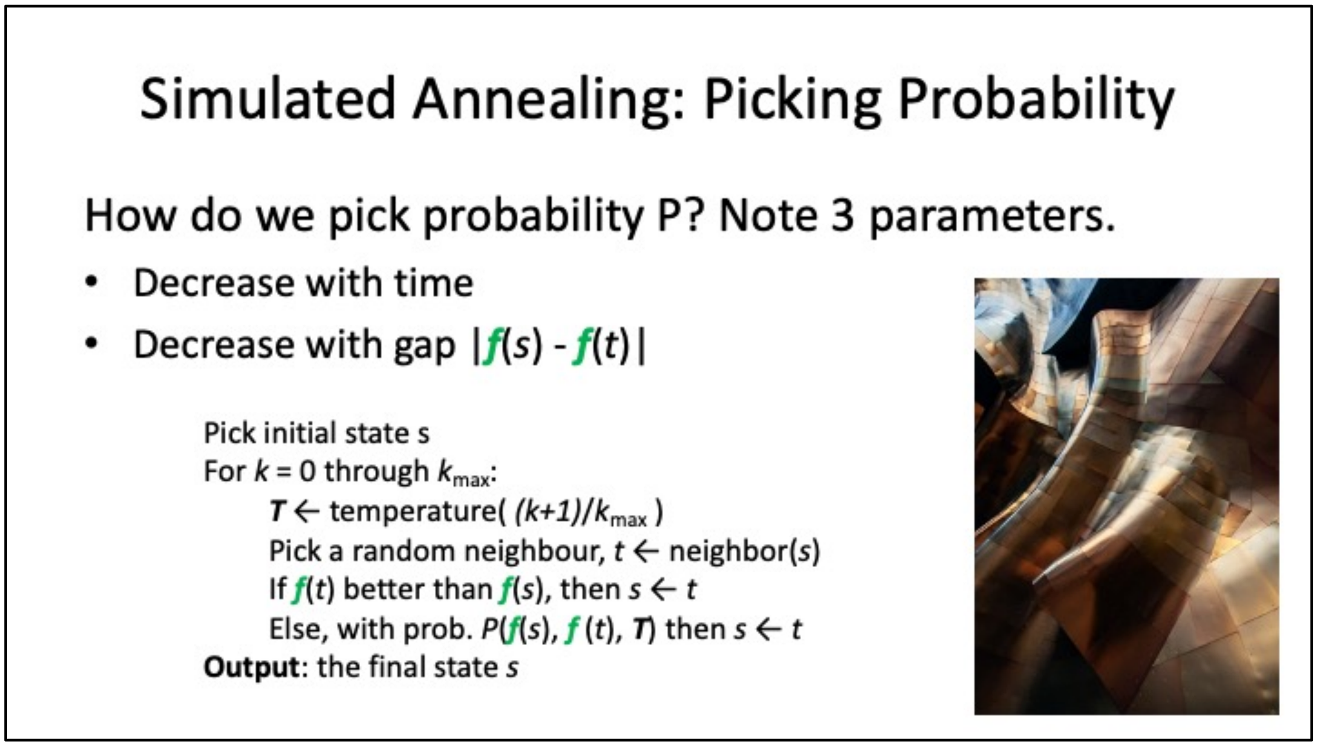
The interesting bit



Can be viewed as a variant of hill climbing: pick neighbors stochastically. But it has a key new idea: at the beginning, the random picking is more relaxed, allowing to pick neighbors more randomly; in the later iterations, the random picking is more greedy, focusing more on good neighbors. The intuition is

1) at the beginning, explore around so as not trapped in poor local optima
2) in the later iterations, begin to settle down on good locations

The selection probability is controlled by a parameter called temperature.

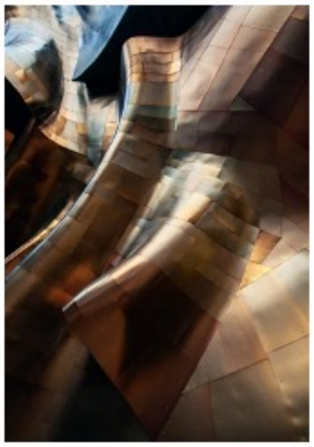


Simulated Annealing: Picking Probability

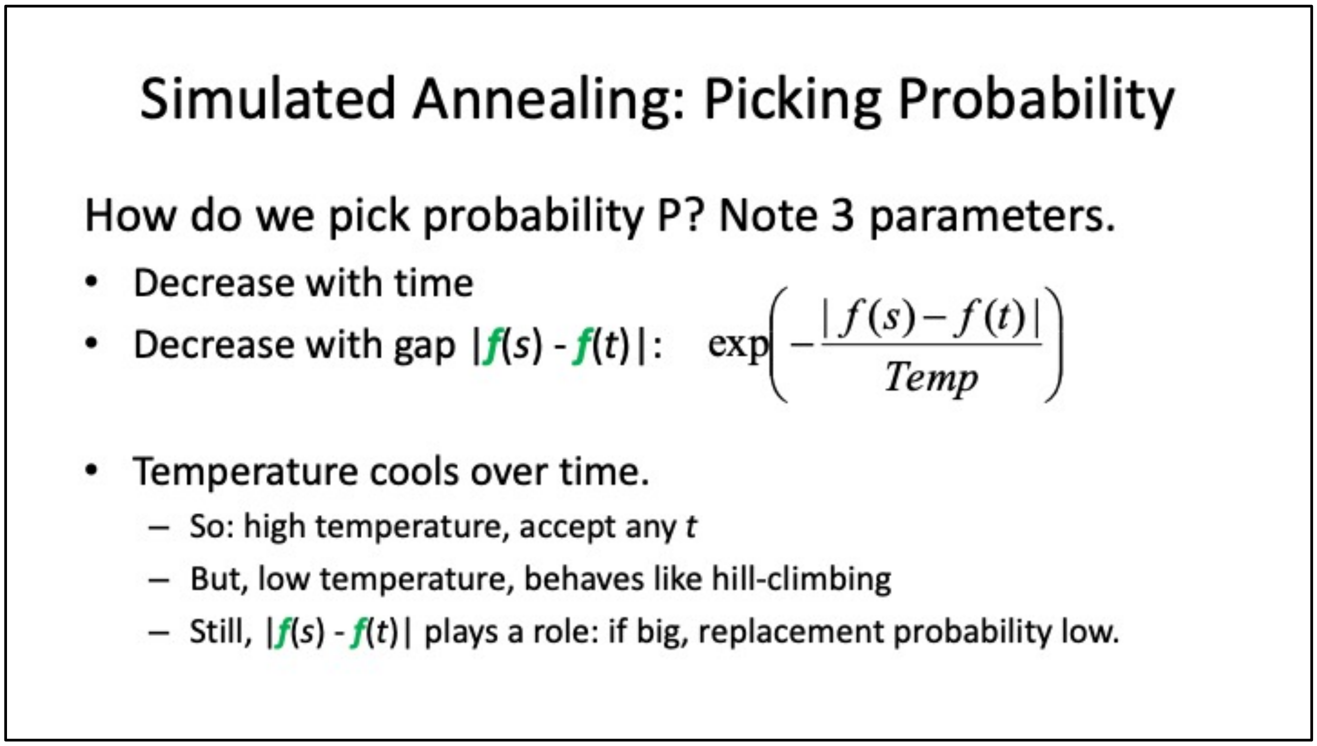
How do we pick probability P? Note 3 parameters.

- Decrease with time
- Decrease with gap $|f(s) - f(t)|$

```
Pick initial state s
For $k$ = 0 through $k_{\max}$:
    $T \leftarrow$ temperature( $(k+1)/k_{\max}$ )
    Pick a random neighbour, $t \leftarrow$ neighbor($s$)
    If $f(t)$ better than $f(s)$, then $s \leftarrow t$
    Else, with prob. $P(f(s), f(t), T)$ then $s \leftarrow t$
Output: the final state $s$
```



The probability is computed based on 3 parameters.
It should have the following properties: decrease with time, and decrease with the gap.



Simulated Annealing: Picking Probability

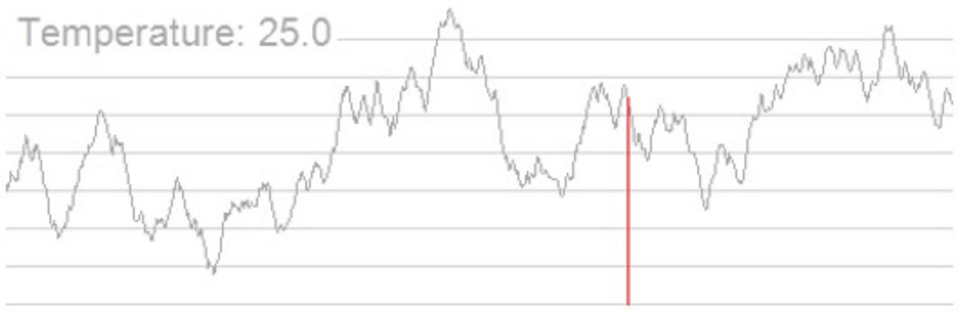
How do we pick probability P? Note 3 parameters.

- Decrease with time
- Decrease with gap $|f(s) - f(t)|$: $\exp\left(-\frac{|f(s)-f(t)|}{Temp}\right)$
- Temperature cools over time.
  - So: high temperature, accept any t
  - But, low temperature, behaves like hill-climbing
  - Still, $|f(s) - f(t)|$ plays a role: if big, replacement probability low.

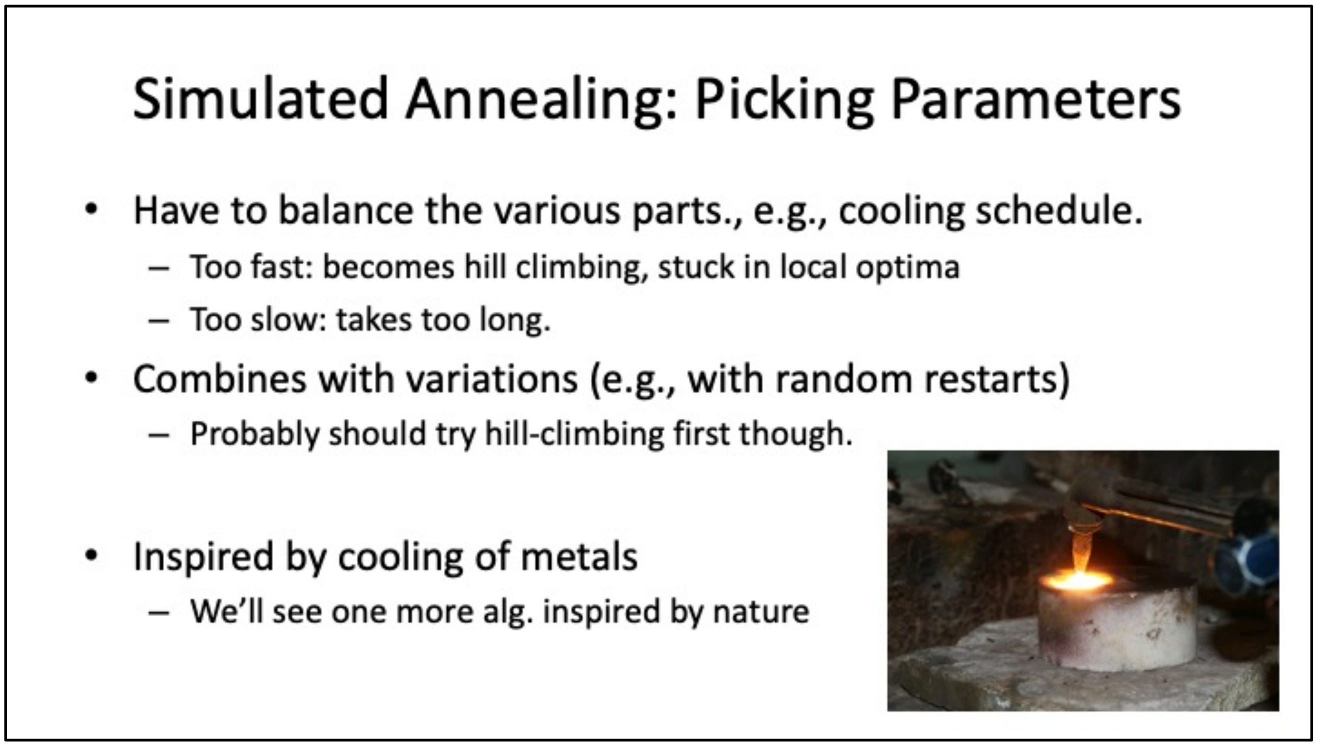
A typical choice is the exponential function.
High temperature: then the probability is close to exp(-0)=1, ie, accept with high probability
Low temperature: if the gap is not very small then the probability will be very small. So only accept those that has value very close to the current solution. Overall the algo looks like hill climbing.

Simulated Annealing: Visualization

What does it look like in practice?



Wiki



Simulated Annealing: Picking Parameters

- Have to balance the various parts., e.g., cooling schedule.
  - Too fast: becomes hill climbing, stuck in local optima
  - Too slow: takes too long.
- Combines with variations (e.g., with random restarts)
  - Probably should try hill-climbing first though.
- Inspired by cooling of metals
  - We'll see one more alg. inspired by nature

Technical comments:

1. Quite a lot of parameters. Need to design the cooling schedule properly.
2. Can be combined with other tricks like random restart.

Break & Quiz

Q 2.1: Which of the following is likely to give the best cooling schedule for simulated annealing?

A. $\text{Temp}_{t+1} = \text{Temp}_t * 1.25$
B. $\text{Temp}_{t+1} = \text{Temp}_t$
C. $\text{Temp}_{t+1} = \text{Temp}_t * 0.8$
D. $\text{Temp}_{t+1} = \text{Temp}_t * 0.0001$

Break & Quiz

Q 2.1: Which of the following is likely to give the best cooling schedule for simulated annealing?

A. $Temp_{t+1} = Temp_t * 1.25$

B. $Temp_{t+1} = Temp_t$

C. $Temp_{t+1} = Temp_t * 0.8$

D. $Temp_{t+1} = Temp_t * 0.0001$

Break & Quiz

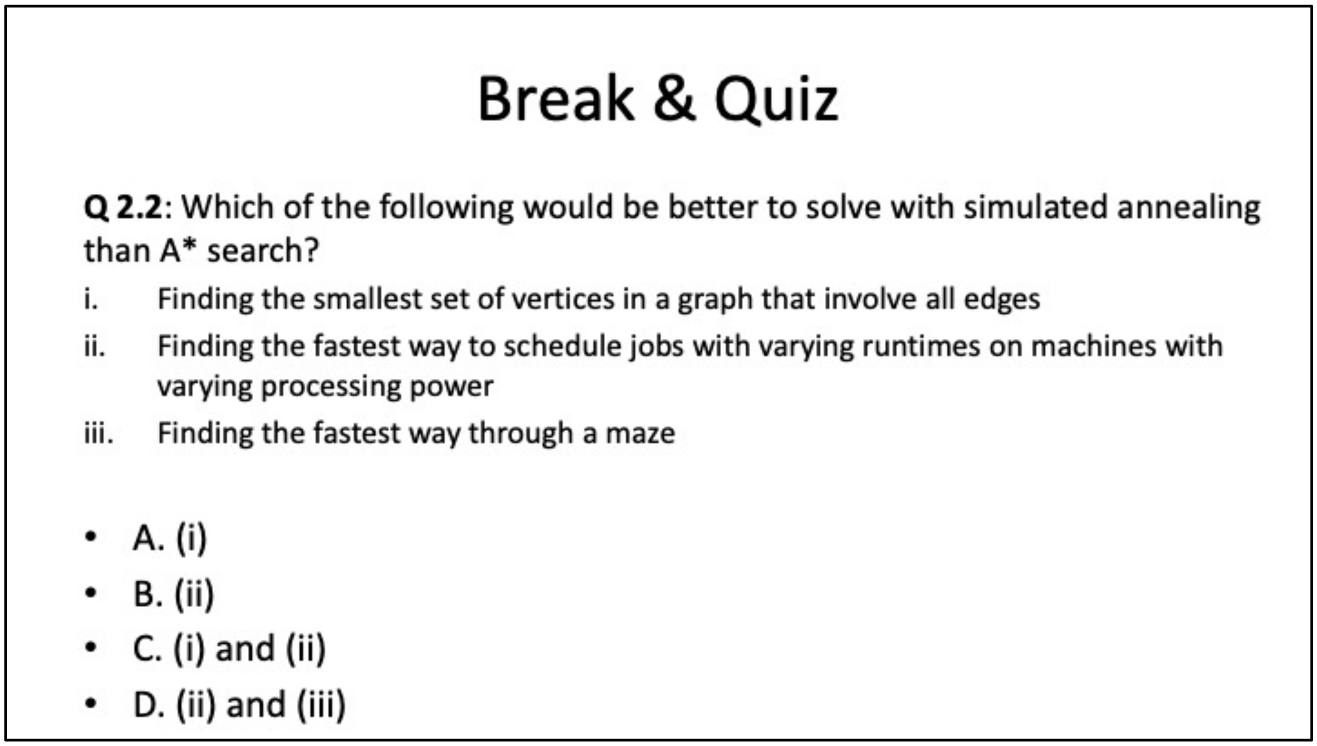
Q 2.1: Which of the following is likely to give the best cooling schedule for simulated annealing?

A. $\text{Temp}_{t+1} = \text{Temp}_t * 1.25$ (No, temperate is increasing)

B. $\text{Temp}_{t+1} = \text{Temp}_t$ (No, temperature is constant)

C. $\text{Temp}_{t+1} = \text{Temp}_t * 0.8$

D. $\text{Temp}_{t+1} = \text{Temp}_t * 0.0001$ (Cools too fast---basically hill climbing)

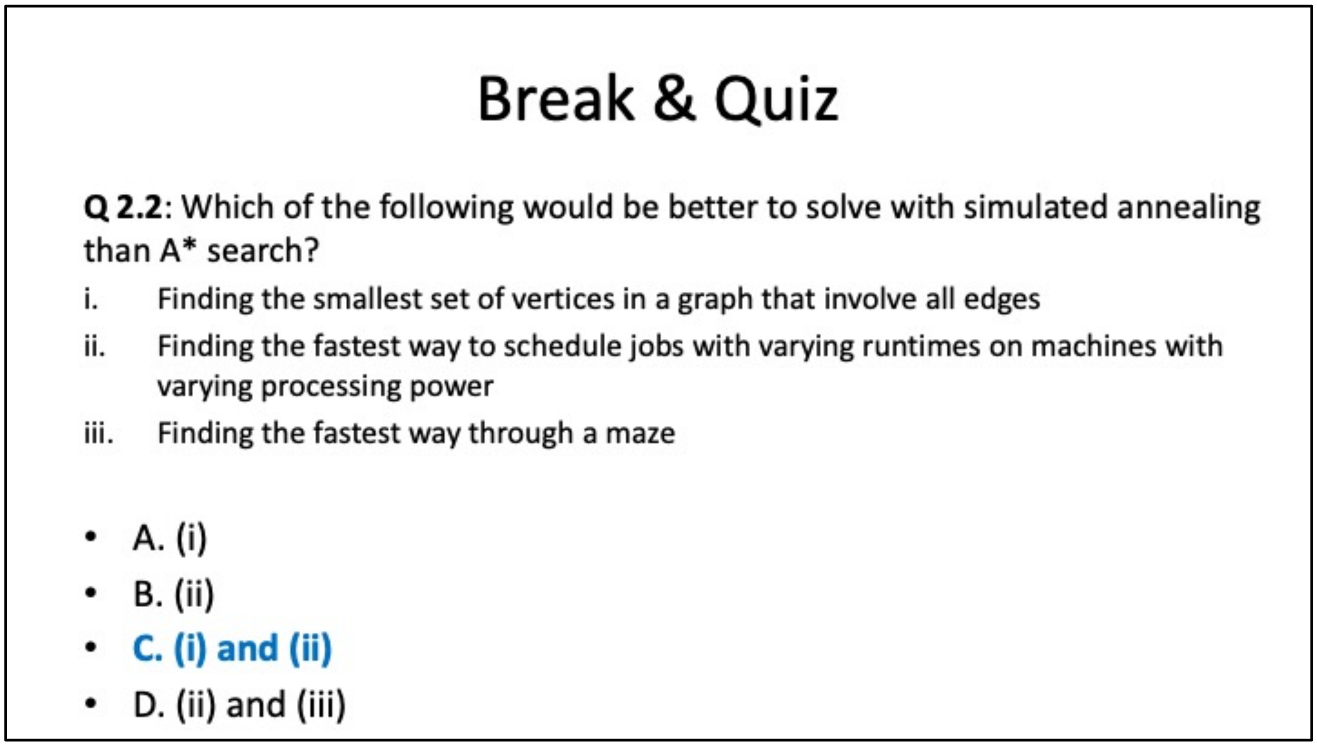


Break & Quiz

Q 2.2: Which of the following would be better to solve with simulated annealing than A* search?

i. Finding the smallest set of vertices in a graph that involve all edges
ii. Finding the fastest way to schedule jobs with varying runtimes on machines with varying processing power
iii. Finding the fastest way through a maze

- A. (i)
- B. (ii)
- C. (i) and (ii)
- D. (ii) and (iii)



Break & Quiz

Q 2.2: Which of the following would be better to solve with simulated annealing than A* search?

i. Finding the smallest set of vertices in a graph that involve all edges
ii. Finding the fastest way to schedule jobs with varying runtimes on machines with varying processing power
iii. Finding the fastest way through a maze

- A. (i)
- B. (ii)
- **C. (i) and (ii)**
- D. (ii) and (iii)

Break & Quiz

Q 2.2: Which of the following would be better to solve with simulated annealing than A* search?

i. Finding the smallest set of vertices in a graph that involve all edges
ii. Finding the fastest way to schedule jobs with varying runtimes on machines with varying processing power
iii. Finding the fastest way through a maze

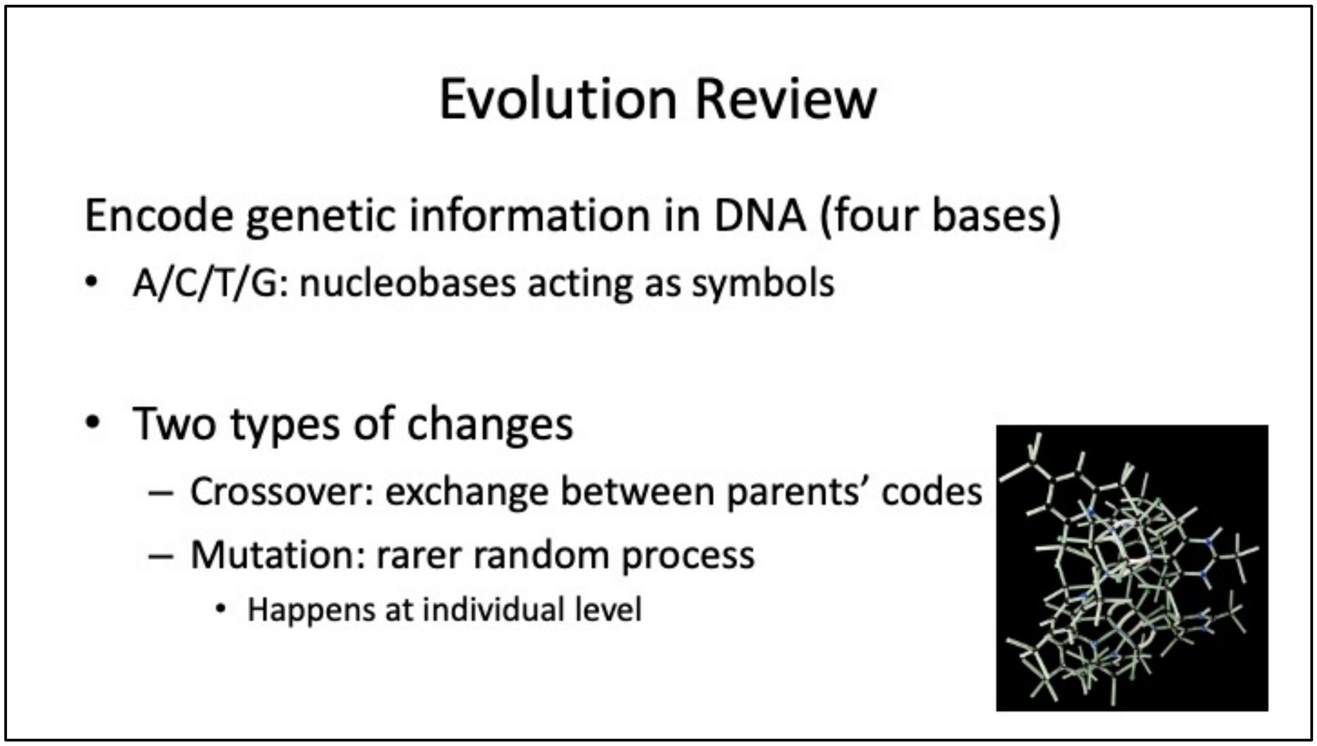
- A. (i) (No, (ii) better: huge number of states, don't care about path)
- B. (ii) (No, (i) complete graph might have too many edges for A*)
- **C. (i) and (ii)**
- D. (ii) and (iii) (No, (iii) is good for A*: few successors, want path)

Genetic Algorithms

Another optimization approach based on nature

- Survival of the fittest!

Genetic algorithms are inspired by evolution: the key idea is that the fittest can survive.

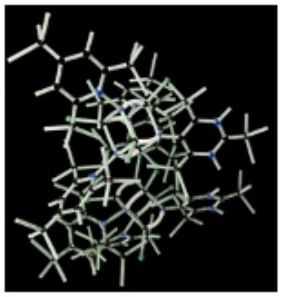


Evolution Review

Encode genetic information in DNA (four bases)

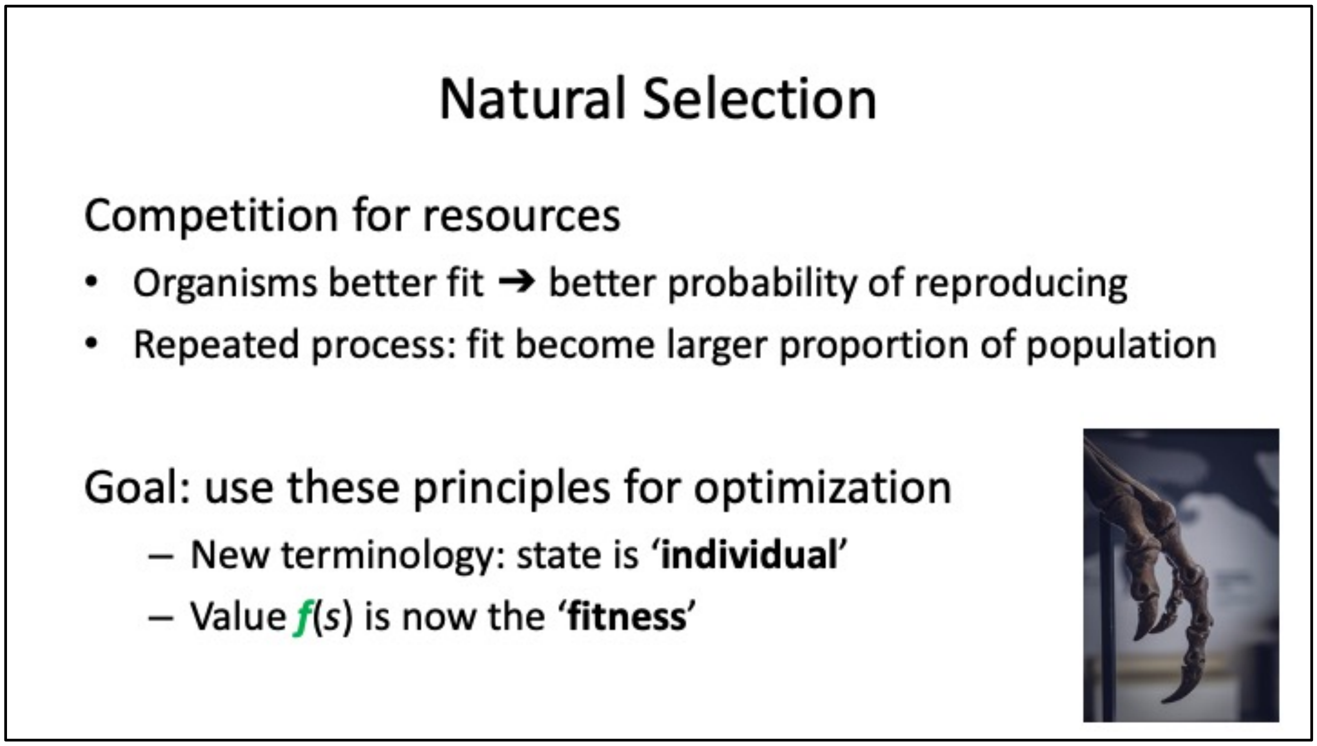
- A/C/T/G: nucleobases acting as symbols

- Two types of changes
  - Crossover: exchange between parents' codes
  - Mutation: rarer random process
    - Happens at individual level



A few key elements about evolution:

1. The genetic information is encoded in DNA, a sequence of four bases: each individual can be viewed as a string of DNA code; a population is viewed as a set of such strings.
2. There are two types of changes on the DNA in evolution: cross-over and mutation



Natural Selection

Competition for resources

- Organisms better fit ➔ better probability of reproducing
- Repeated process: fit become larger proportion of population

Goal: use these principles for optimization

- New terminology: state is **'individual'**
- Value $f(s)$ is now the **'fitness'**

Another key element in evolution is natural selection induced by competition for resources: The DNA that fits better the environment has better chance of survival and reproduction.
The natural selection process is repeated in generations. After multiple generations, a large fraction of the population will be those that fit the environment.

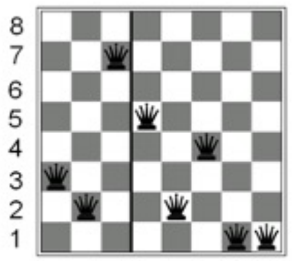
Here we use these principles to design an optimization method. Now the states in the state space now correspond to individuals, and the value function f corresponds to a measurement of fitness. The larger value for f, the better the individual fit the environment. Then we can use the evolution to maximize the value f by finding the best fit.

Genetic Algorithms Setup I

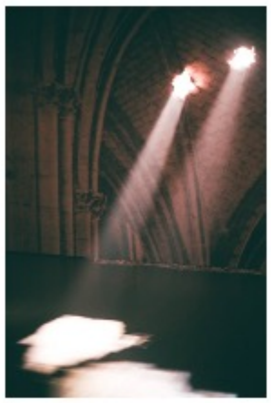
Keep around a fixed number of states/individuals

- Call this the **population**

For our n Queens game example, an individual:



(3 2 7 5 2 4 1 1)

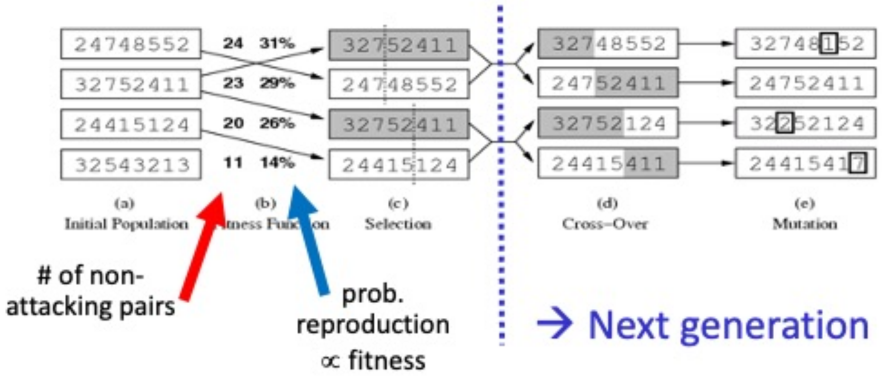


Example:
First we need to encode the state into a string. For n Queens we can encode it as the string of the positions of the queens in each column.
Then we keep a population, a set of individuals.

Genetic Algorithms Setup II

Goal of genetic algorithms: optimize using principles inspired by mechanism for evolution

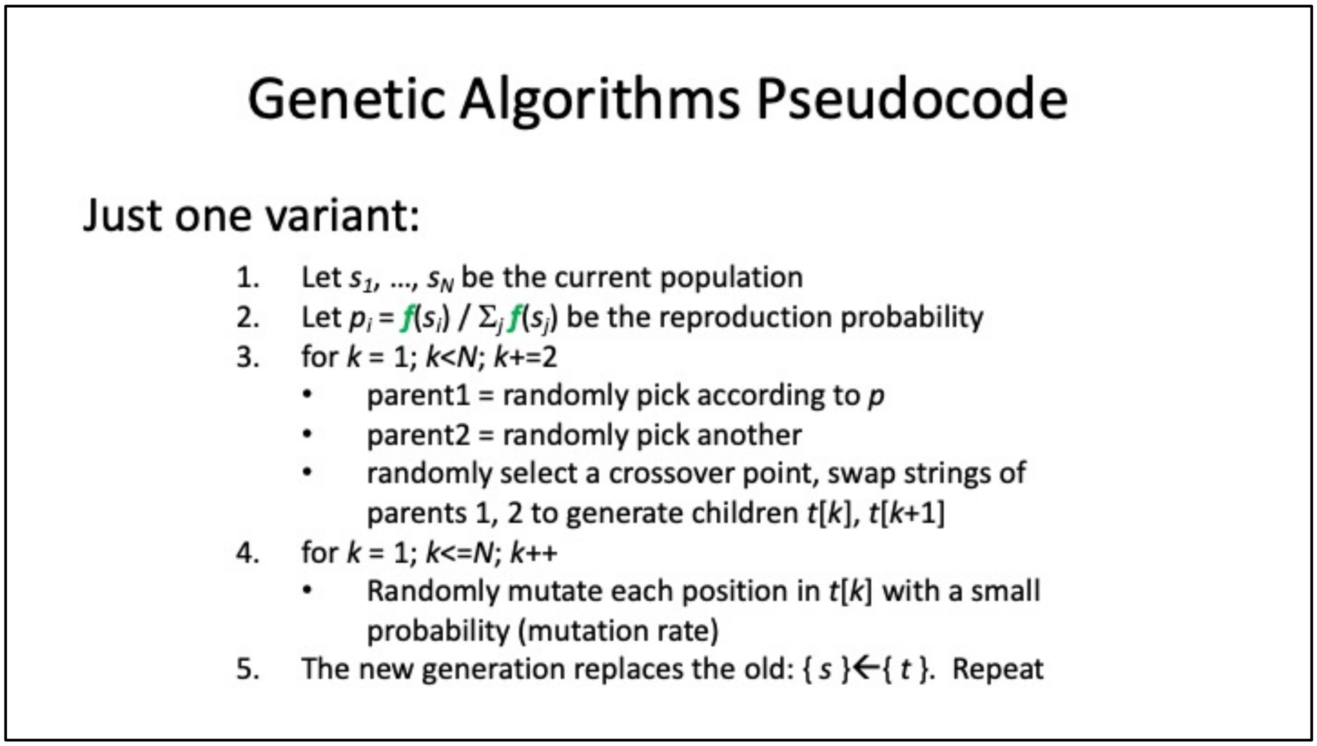
- E.g., analogous to **natural selection, cross-over**, and **mutation**



The genetic algorithm runs in generations, each generation has the following operations corresponding to natural selection, crossover and mutation.

1. Compute the f value (the fitness), and then normalize them to get the probability of reproduction. Then do natural selection: by sampling from the reproduction probability distribution.
2. Then pair up the individuals; for each pair, pick a random location to cut each code into two segments; then do crossover by mixing up the segments.
3. Finally do mutation: (one standard variant) for each location in each individual, determine whether to do mutation with a small mutation probability; if yes, then replace the original symbol with a randomly pick symbol.

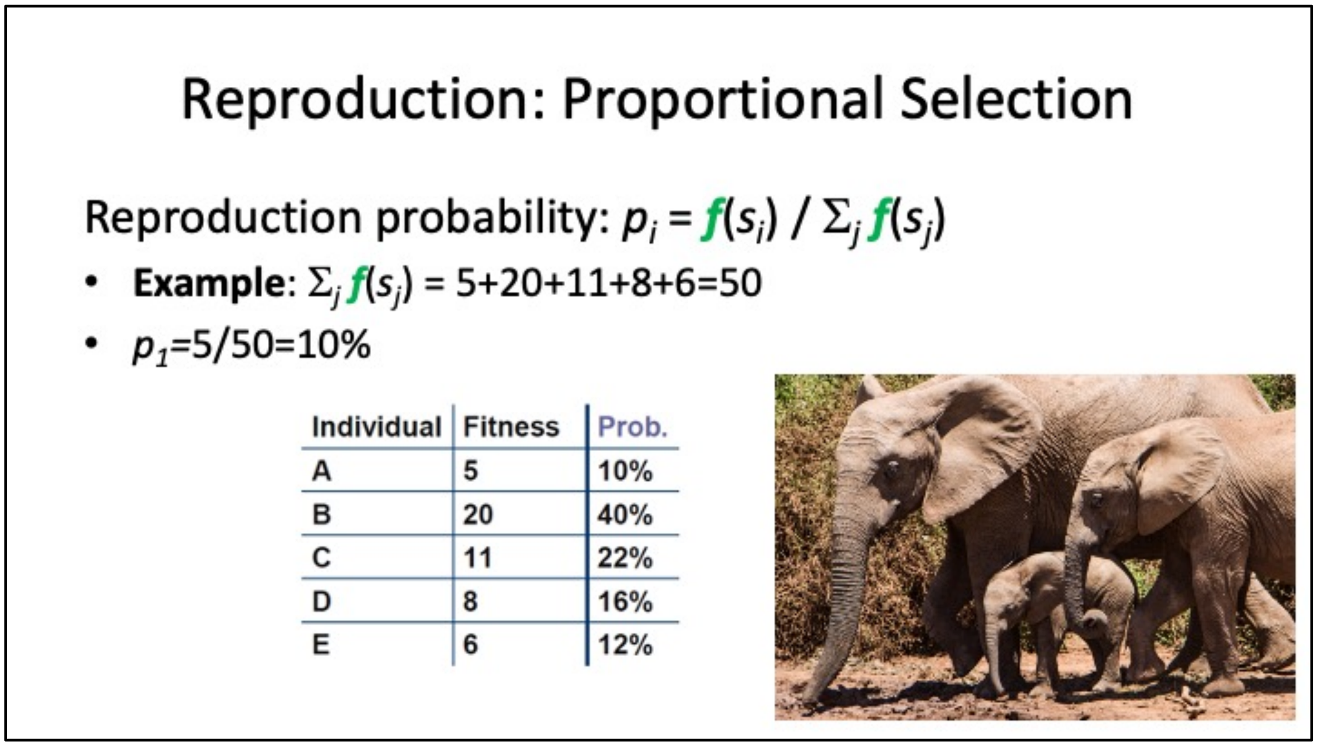
Repeat this until we are satisfied with the solution or run out of time budget.



Genetic Algorithms Pseudocode

Just one variant:

```
1. Let $s_1, \ldots, s_N$ be the current population
2. Let $p_i = f(s_i) / \Sigma_j f(s_j)$ be the reproduction probability
3. for $k = 1$; $k<N$; $k$+=2
     • parent1 = randomly pick according to $p$
     • parent2 = randomly pick another
     • randomly select a crossover point, swap strings of
       parents 1, 2 to generate children $t[k]$, $t[k+1]$
4. for $k = 1$; $k<=N$; $k$++
     • Randomly mutate each position in $t[k]$ with a small
       probability (mutation rate)
5. The new generation replaces the old: $\{ s \} \leftarrow \{ t \}$.  Repeat
```



Reproduction: Proportional Selection

Reproduction probability: $p_i = \boldsymbol{f}(s_i) \,/\, \Sigma_j \boldsymbol{f}(s_j)$

- **Example**: $\Sigma_j \boldsymbol{f}(s_j) = 5+20+11+8+6=50$
- p_1=5/50=10%

Individual	Fitness	Prob.
A	5	10%
B	20	40%
C	11	22%
D	8	16%
E	6	12%

An extra example of computing the reproduction probability: normalizing the fitness score.

Acknowledgements: Adapted from materials by Jerry Zhu + Tony Gitter (University of Wisconsin), Andrew Moore