Class 5 - Introduction to Genetic Algorithms

Guiding Genetic Algorithms with Language Models for Combinatorial Optimization

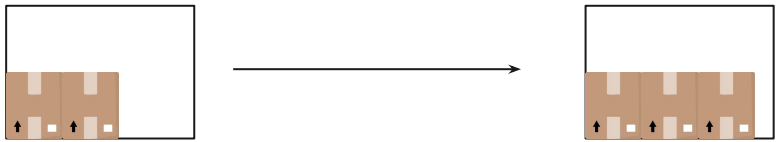
MISE Research Program - July 2025

Pedro Paredes

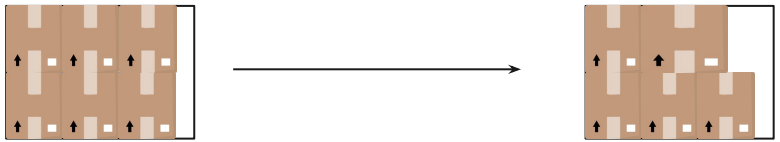
Hill Climbing

Search or Optimization

In **search algorithms** (e.g. Exhaustive Search, Informed Search, A*) states are partial solutions that we gradually build up into complete solutions



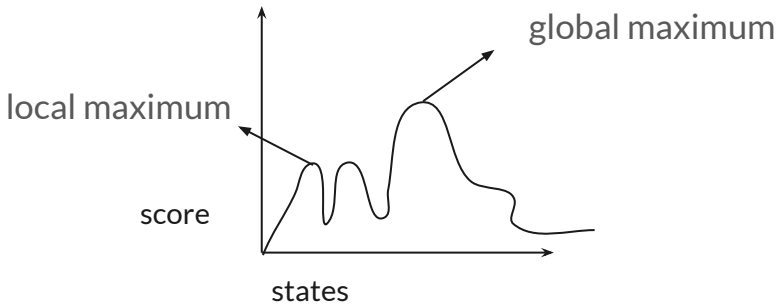
In **optimization algorithms** (e.g. Hill Climbing, Genetic Algorithms) states are full solutions that we move between



Finding Maximums in Functions

The ***state space*** of a problem is the set of all possible states, e.g., for N-Queens are boards with one queen per column

The problems we have been thinking about can be modeled as finding the **maximum of a score function** over a state space



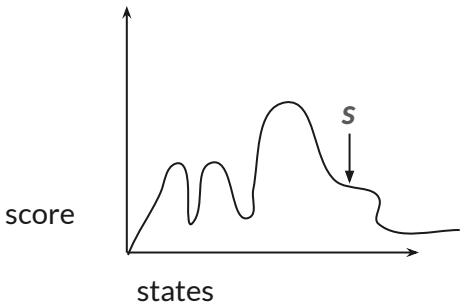
Hill Climbing

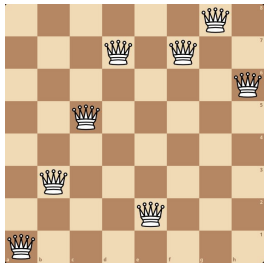
Hill climbing is a method to find a solution to a problem by always moving in the direction of a “better” solution

- Start from a random state ***s***
- Move to another state with better score
- If we get stuck and didn’t find a solution, restart

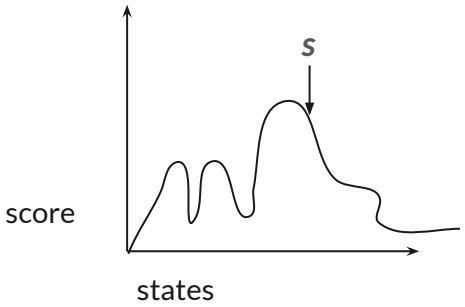
Hill Climbing

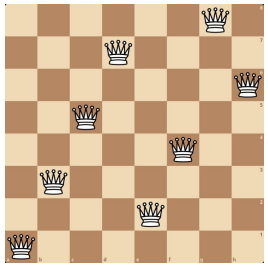
Pick a random state ***s***



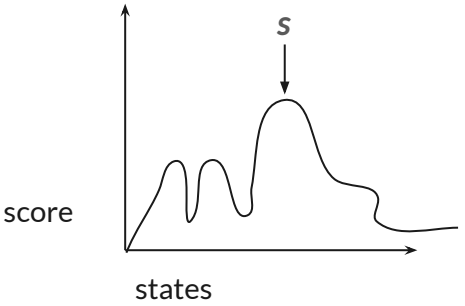


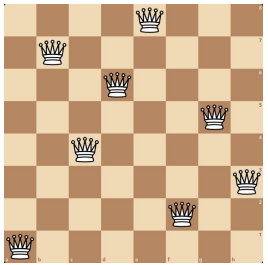
Move to a better state





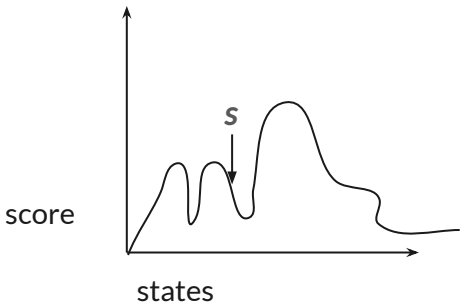
Repeat and find optimal solution

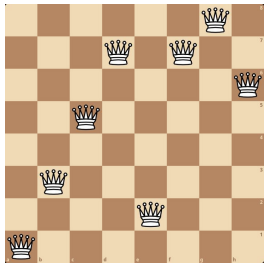




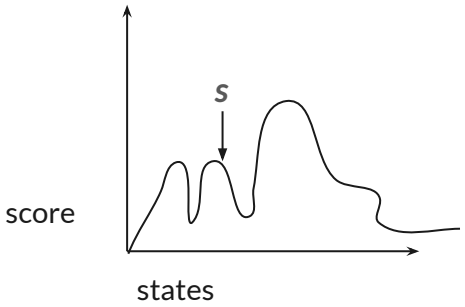
Hill Climbing

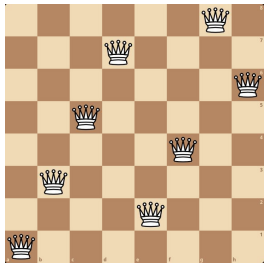
Pick a random state ***s***



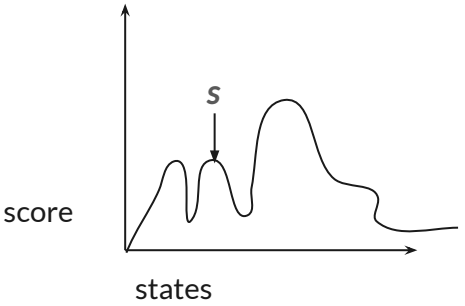


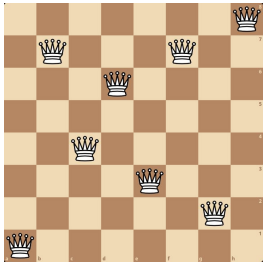
Move to a better state





Get stuck, so restart





Designing a Hill Climbing Method

The only design choice in a Hill Climbing method is how to move between states

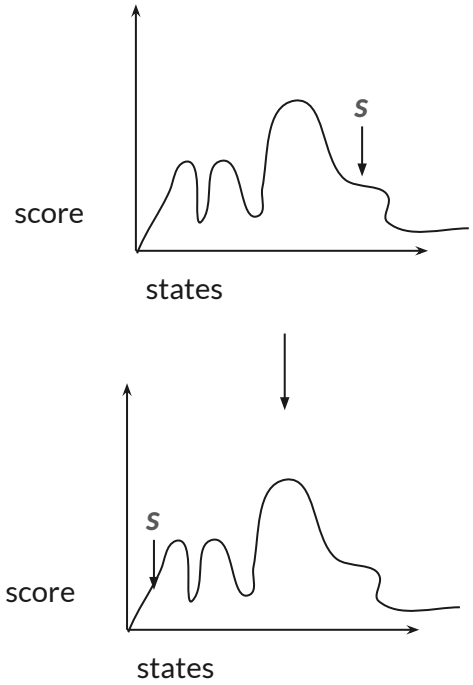
How should we move to another state?

- Define a **local neighborhood** of the state
- Pick the best state in the local neighborhood (break ties randomly)

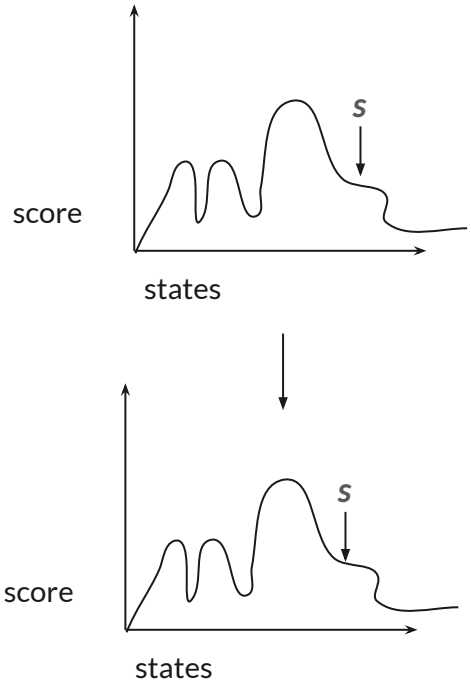
How should we define a local neighborhood?

- If it's too small: takes a long time to converge to maximum
- If it's too big: hard to converge to maximum because of too much randomness

Defining Local Neighborhoods



Neighborhood too large



Neighborhood too small

Hill Climbing for N-Queens

States: boards with one queen per column

Score function of a state: number of pairs of queens attacking each other (i.e., conflicts)

Transitions: pick one column and move that queen to a different row in that column

Genetic Algorithms

Genetic Algorithm

A ***Genetic Algorithm (GA)*** is a method inspired by the evolutionary process of “survival of the fittest”

1. Generate **P** random states -> call them your population
2. Repeat until optimal found: (***evolution step***)
    a. Pick **C** pairs of states and randomly mix them (***crossover***)
    b. For each new state, with some small probability **p** apply a local change (***mutation***)
    c. Discard all but the **P** states with best score (***selection***)

Some variations:

- Keep the top x% of the parent population and top (100-x)% of the new population
- The values of **P**/**C**/**p** evolve throughout the algorithm (also known as adaptive genetic algorithm)

Population

A ***population*** is a collection of **P** states, where **P** is a parameter you can choose

How should we pick **P**?

- If it's too small: each evolution step improves a small amount
- If it's too big: making one evolution step takes too long

The population will evolve throughout the execution of the GA

Crossover

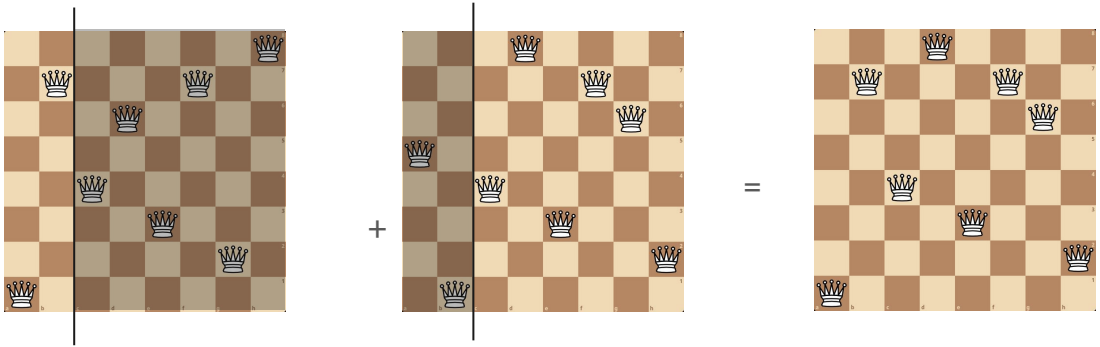
Crossover is an operation that takes two states and combines them by taking some features of one state and some features of the other

We pick **C** pairs of states from the current population to crossover, where **C** is a parameter you choose

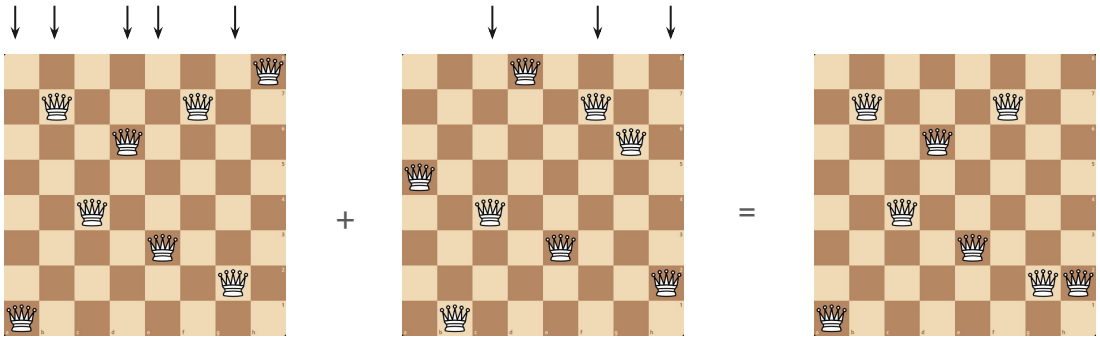
A few common types of crossover:

- One-point crossover: pick one dividing point and take the first half of one state and the second of the other
- Uniform crossover: take each element uniformly at random from each state

Crossover - N-Queens Example



one-point crossover



uniform crossover

Crossover - Reproduction Probability

We want the most promising states to prevail, so we want to apply crossovers to them more often

The ***reproduction probability*** of a state is given by the score of the state divided by the total sum of the scores of all other states in the population: *probability(**s**) = f(**s**) / sum(f(**os**) for **os** in **population**)*

To pick two states to crossover, first pick one according to the reproduction probability, and then pick the second one uniformly at random

We can use the method `random.choices` to choose an element from a list according to a probability

Mutation

Mutation is an operation that applies one local transformation with some probability **p**

A local transformation is a random state from a local neighborhood (as defined for Hill Climbing)

How should we pick **p**?

- If it's too small: not enough randomness means we can get stuck near local maximums
- If it's too big: hard to converge to maximum because of too much randomness

Genetic Algorithm - Recap

1. Generate **P** random states -> call them your population
2. Repeat until optimal found: (***evolution step***)
    a. Pick **C** pairs of states and randomly mix them (***crossover***)
    b. For each new state, with some small probability **p** apply a local change (***mutation***)
    c. Discard all but the **P** states with best score (***selection***)

What's next?

Next class wednesday (3pm)

Assignment sheet out soon, contains one large problem -> email me your solution when done

Class 6: Implementing a Genetic Algorithm in Python