CSC520 - Artificial Intelligence
Lecture 6

Dr. Scott N. Gerard

North Carolina State University

Jan 28, 2025

Agenda

- Local search
  - Hill climbing
  - Simulated annealing
  - Local beam search
  - Genetic algorithms

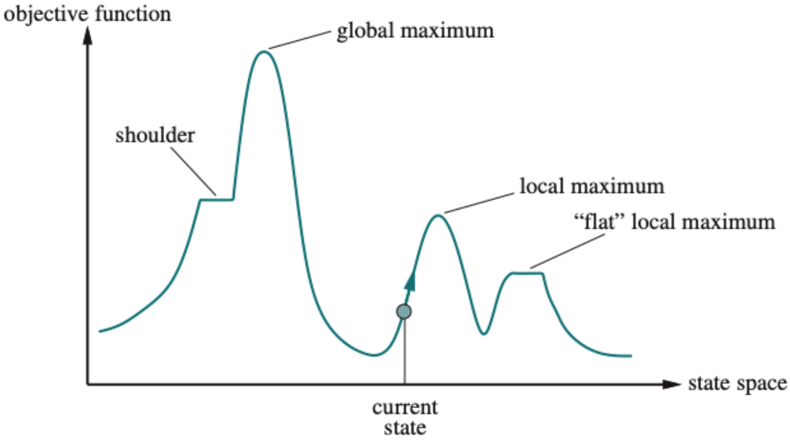
Local Search

- Previous search algorithms produce a *path* to the goal as a solution
  - Keep paths in memory
  - Remember alternative paths for backtracking
- In some problems, final state is solution. The path is irrelevant
  - 8-queens
  - Factory layout
  - IC design
- Local search algorithms produce the final state as the solution

Local Search and Optimization

- From a state, explore the neighboring states
- Does not keep track of paths or states that have been reached
- Uses very little memory
- Can often find reasonable solutions in large or infinite state spaces
- Aims to maximize (or minimize) an objective function defined on the state space
- Start state may not be specified

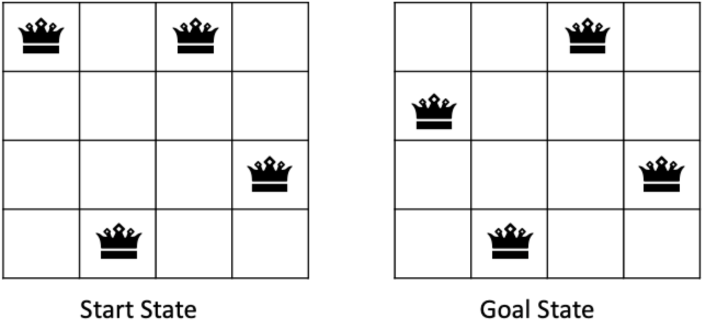
Optimization



Hill-climbing Search

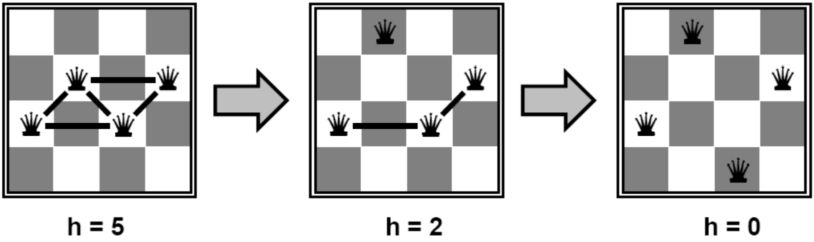
function HILL-CLIMBING(*percept*) **return** a state
    current $\leftarrow$ *initial_state*
    while true **do**
        neighbor $\leftarrow$ highest-valued successor state of current
        if VALUE(*neighbor*) $\leq$ VALUE(*current*) **then**
            return *current*
        current $\leftarrow$ *neighbor*

4-Queen Problem



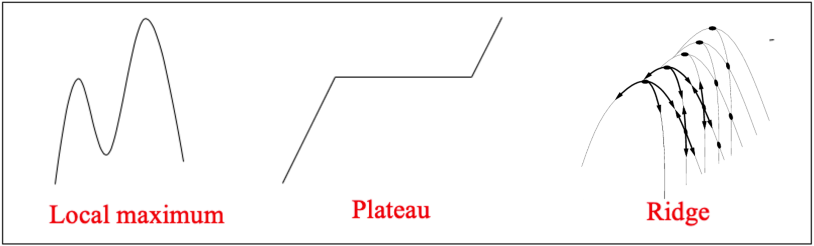
4-Queen Problem

- Objective function: Number of pairwise conflicts
- Aim is to minimize the objective function
- Move one queen within its column to reduce conflicts



Problems with Hill-climbing Search

- Local maxima: a peak that is lower than the highest peak
- Plateaus: flat area of the objective function landscape
- Ridges: sequence of local maximas



Problems with Hill-climbing Search

h = 17

- $8^8 \approx 17$ million states
- Hill-climbing solves 14% of the problem instances taking only 4 steps on average
- Gets stuck 86% of time at a local minimum
- With sideways move (same h) to escape plateau, hill-climbing solves 94% of the problem instances, but with 21 steps on average

Hill-climbing Improvements

- Stochastic hill-climbing
  - Random selection among the uphill moves
  - Selection probability proportional to the uphill steepness

Hill-climbing Improvements

- Stochastic hill-climbing
  - Random selection among the uphill moves
  - Selection probability proportional to the uphill steepness
- Random-walk hill-climbing
  - Greedy: With probability p, move to a neighbor with largest value
  - Random: With probability $1 - p$, move to a random neighbor

Hill-climbing Improvements

- Stochastic hill-climbing
  - Random selection among the uphill moves
  - Selection probability proportional to the uphill steepness
- Random-walk hill-climbing
  - Greedy: With probability p, move to a neighbor with largest value
  - Random: With probability $1 - p$, move to a random neighbor
- Random-restart hill-climbing
  - Run hill-climbing multiple times with a randomly generated initial state

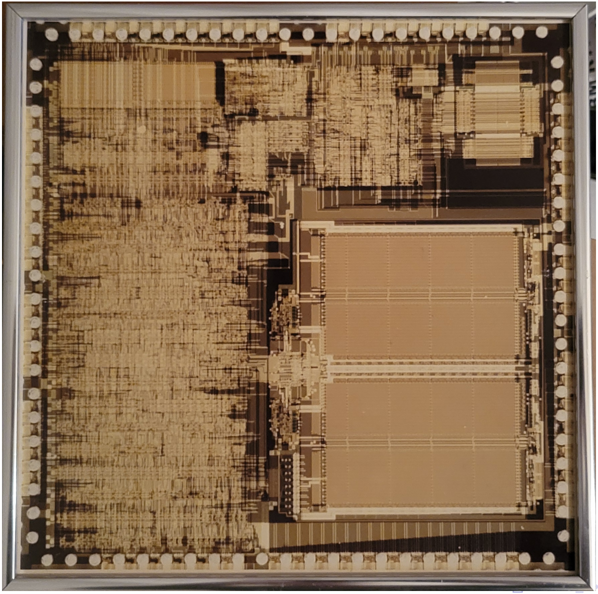
Hill-climbing Improvements

- Stochastic hill-climbing
  - Random selection among the uphill moves
  - Selection probability proportional to the uphill steepness
- Random-walk hill-climbing
  - Greedy: With probability p, move to a neighbor with largest value
  - Random: With probability $1 - p$, move to a random neighbor
- Random-restart hill-climbing
  - Run hill-climbing multiple times with a randomly generated initial state
- Hill-clibming with both random-walk and random-restart

Simulated Annealing

- Inspired by *annealing* process in metallurgy used to harden metals
- A temperature variable is set to a high initial value and decreased in each iteration
- Similar to hill-climbing but instead of picking the *best* move, it picks a *random* move
- If the move improves the objective function, it is accepted
- Else accept the move with a probability that decreases with the badness of the move and with the temperature
- If temperature is decreased slowly enough, the algorithm will find global maxima

Simulated Annealing



Simulated Annealing

function $\textsc{Simulated-Annealing}(\mathit{problem}, \mathit{schedule})$ **return** a state

$\mathit{current} \leftarrow \mathit{problem.initial\_state}$

for $t = 1$ to ∞ **do**

$T \leftarrow \mathit{schedule}(t)$

if T $= 0$ **then return** *current*

next $\leftarrow$ random successor of current

$\Delta E \leftarrow \textsc{Value}(\mathit{current}) - \textsc{Value}(\mathit{next})$

if $\Delta E > 0$ **then** $\mathit{current} \leftarrow \mathit{next}$

else $\mathit{current} \leftarrow \mathit{next}$ with probability $e^{\Delta E/T}$

Local Beam Search

- Idea: Instead of keeping just 1 state, keep k best states
- Start with k randomly selected states
- Generate successors of the k states
- If any of the successors is goal, then return that successor
- Else select k best states from the successors and repeat

Local Beam Search

- Idea: Instead of keeping just 1 state, keep k best states
- Start with k randomly selected states
- Generate successors of the k states
- If any of the successors is goal, then return that successor
- Else select k best states from the successors and repeat
- Problem: all k states end up on the same local maxima
- Stochastic beam search addresses the problem by choosing k successors randomly biased toward good ones

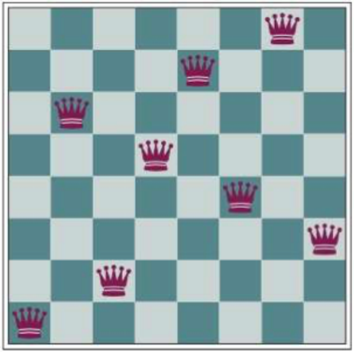
Local Beam Search

- Idea: Instead of keeping just 1 state, keep k best states
- Start with k randomly selected states
- Generate successors of the k states
- If any of the successors is goal, then return that successor
- Else select k best states from the successors and repeat
- Problem: all k states end up on the same local maxima
- Stochastic beam search addresses the problem by choosing k successors randomly biased toward good ones
- Similar to natural selection: successors (offspring) of a state (organism) populate the next generation according to its value (fitness)

Genetic Algorithms

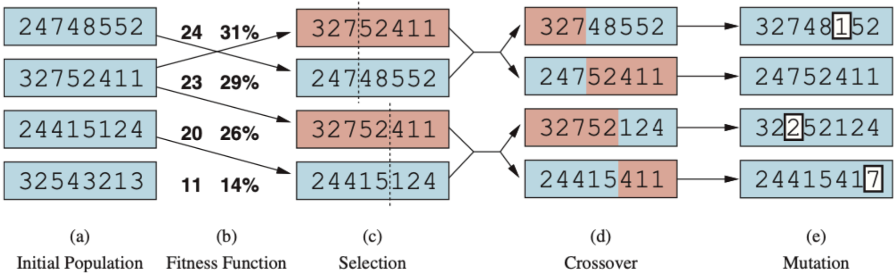
- Variant of stochastic beam search where successors are generated by combining two parents
- A state (genome) is represented as a string over a finite alphabet
- Start with k randomly generated states (population)
- An evaluation or fitness function for a state (phenome)
  - fn(genome) $\Rightarrow$ phenome
- Create next generation of states by simulated evolution
  - Randomly select parents with probability proportional to their fitness
  - Combine parents using a random crossover point
  - Randomly mutate the offspring with probability equal to mutation rate

8-Queen Genetic Algorithm

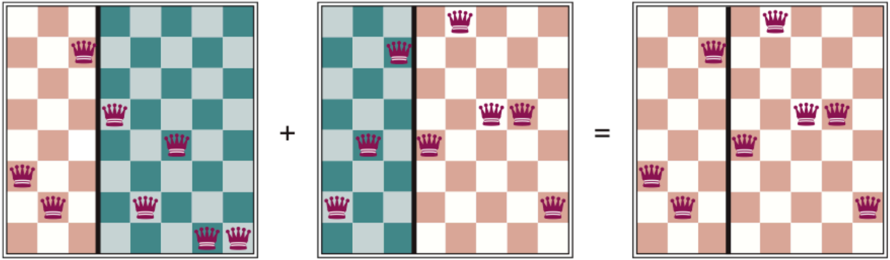


- State (genome) is 8-digit string: each digit represents row number of a queen
- State = 16257483
- Phenome is a chess board with 8 queens.
- Fitness function is the number of *non − attacking* pairs
- Fitness = 27

8-Queen Genetic Algorithm



8-Queen Genetic Algorithm



Genetic Algorithms

- Random exploration can find solutions that local search cannot
- Appealing connection to human evolution (survival of the fittest)

Genetic Algorithms

- Random exploration can find solutions that local search cannot
- Appealing connection to human evolution (survival of the fittest)
- Too many tunable parameters
- Difficult to replicate performance from one problem to another
- Lack of empirical studies comparing to simpler methods
- No convincing evidence that genetic algorithms are better than hill-climbing with random restarts in general