UNIVERSITÀ DEGLI STUDI DI NAPOLI
PARTHENOPE

Artificial Intelligence

Search in Complex Environments

LESSON 6

prof. Antonino Staiano

M.Sc. In ''Machine Learning e Big Data'' - University Parthenope of Naples

Informed Search

- Uniformed and informed search concern with finding a solution as a sequence of actions
  - The environments are fully observable, deterministic, static, and known
- Now, we want to relax some of those constraints
  - Finding a good state without considering the path to get there
    - Discrete and continuous states

Lecture outline

- Local Search and Optimization Problems
  - Hill-climbing
  - Simulated annealing
  - Genetic algorithms
- Local search in continuous spaces

Optimization Problems

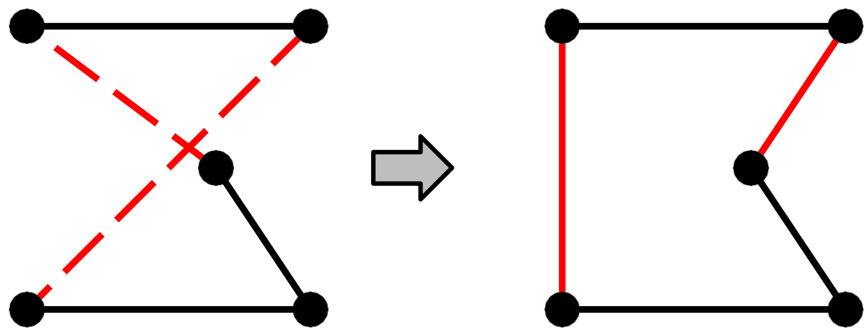
- In search problems examined so far, the agent needed to find a path from a source to a destination
    - A path from Arad to Bucharest
- In many optimization problems, the path is irrelevant
  - The goal state itself is the solution
    - We care only about finding a valid final configuration
      - 8-queens
      - Integrated-circuits design
      - Job shop scheduling
      - Telecommunications network optimization
      - Crop planning

Local Search and Optimization Problems

- The state space is a set of "complete" configurations
  - find the optimal configuration, e.g., TSP
  - find configuration satisfying constraints, e.g., timetable
- In such cases, one can use iterative improvement algorithms
  - keep a single "current" state and try to improve it
- Local search algorithms operate by searching from a start state to neighboring states, without keeping track of the paths, nor the set of states that have been reached
- They are not systematic; they might never explore a portion of the search space where a solution actually resides

Example: TSP

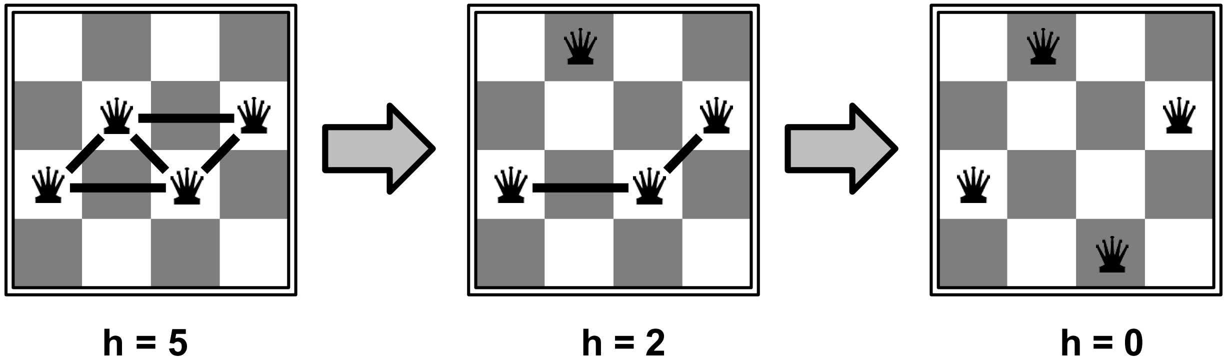
- Start with any complete tour, perform pairwise exchange



- Variants of this approach get within 1% of optimal very quickly with thousands of cities

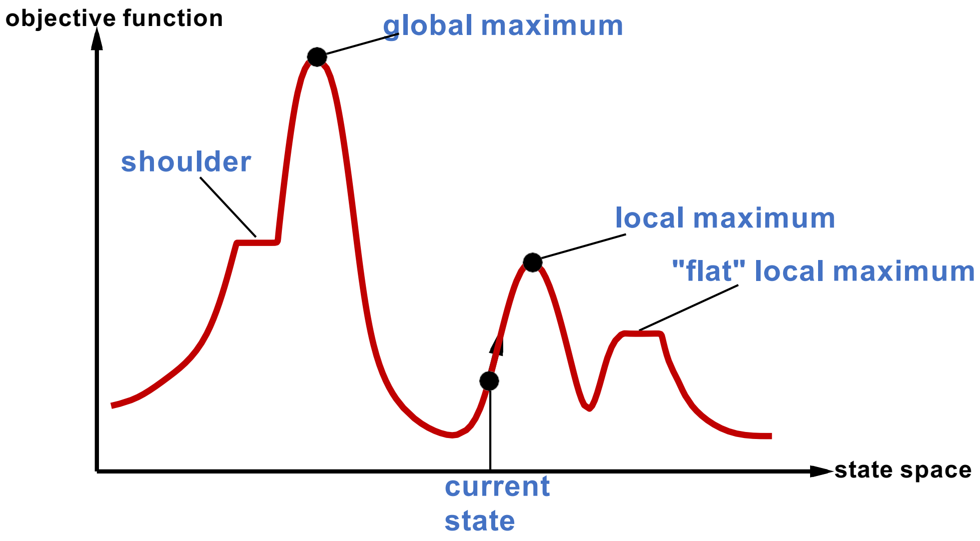
Example: n-Queens

- Put n-queens on a nxn board, with no queen on the same row, column or diagonal
- Move a queen to reduce the number of conflicts



Local Search and Optimization Problems

- Local search algorithms solve optimization problems
  - Find the best state according to an objective function
- Let's consider a state-space landscape

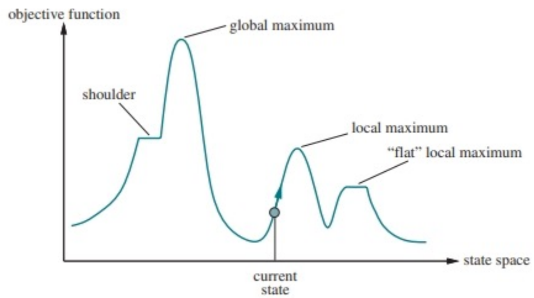


Hill-climbing Search

- It keeps track of one current state and on each iteration moves to the neighboring state with the highest value
  - It heads in the direction that provides the steepest ascent
  - Stops at a peak with no neighbor with a higher value
  - Does not look ahead beyond the immediate neighbors of the current state

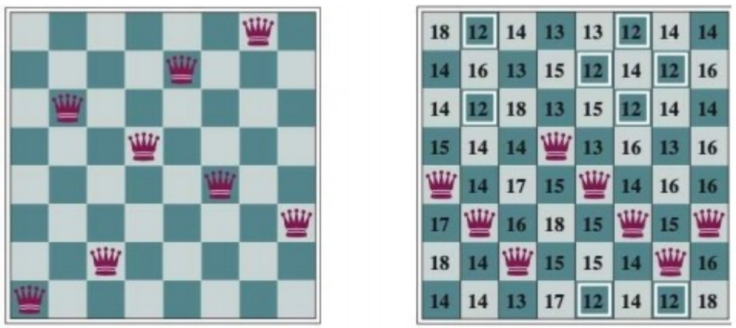
```
function HILL-CLIMBING(problem) returns a state that is a local maximum
  current ← problem.INITIAL
  while true do
    neighbor ← a highest-valued successor state of current
    if VALUE(neighbor) ≤ VALUE(current) then return current
    current ← neighbor
```

Figure 4.2 The hill-climbing search algorithm, which is the most basic local search technique. At each step the current node is replaced by the best neighbor.



Hill-climbing Search and 8-Queens

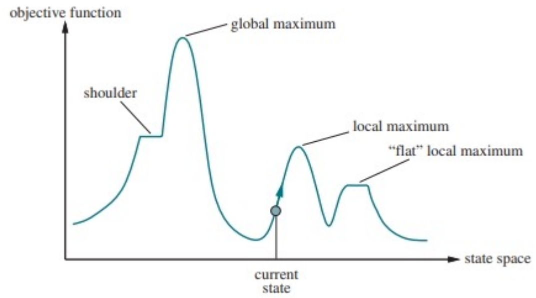
- The initial state is chosen at random
- The successors of a state are all possible states generated by moving one queen to another square in the same column (56 successors)
- The heuristic cost function h is the number of pairs of queens that are attacking each other
  - Zero only for solutions
  - Count as an attack if two pieces in the same line, with an intervening piece between them



(a) (b)

Figure 4.3 (a) The 8-queens problem: place 8 queens on a chess board so that no queen attacks another. (A queen attacks any piece in the same row, column, or diagonal.) This position is almost a solution, except for the two queens in the fourth and seventh columns that attack each other along the diagonal. (b) An 8-queens state with heuristic cost estimate $h = 17$. The board shows the value of h for each possible successor obtained by moving a queen within its column. There are 8 moves that are tied for best, with $h = 12$. The hill-climbing algorithm will pick one of these.

Properties of Hill-climbing



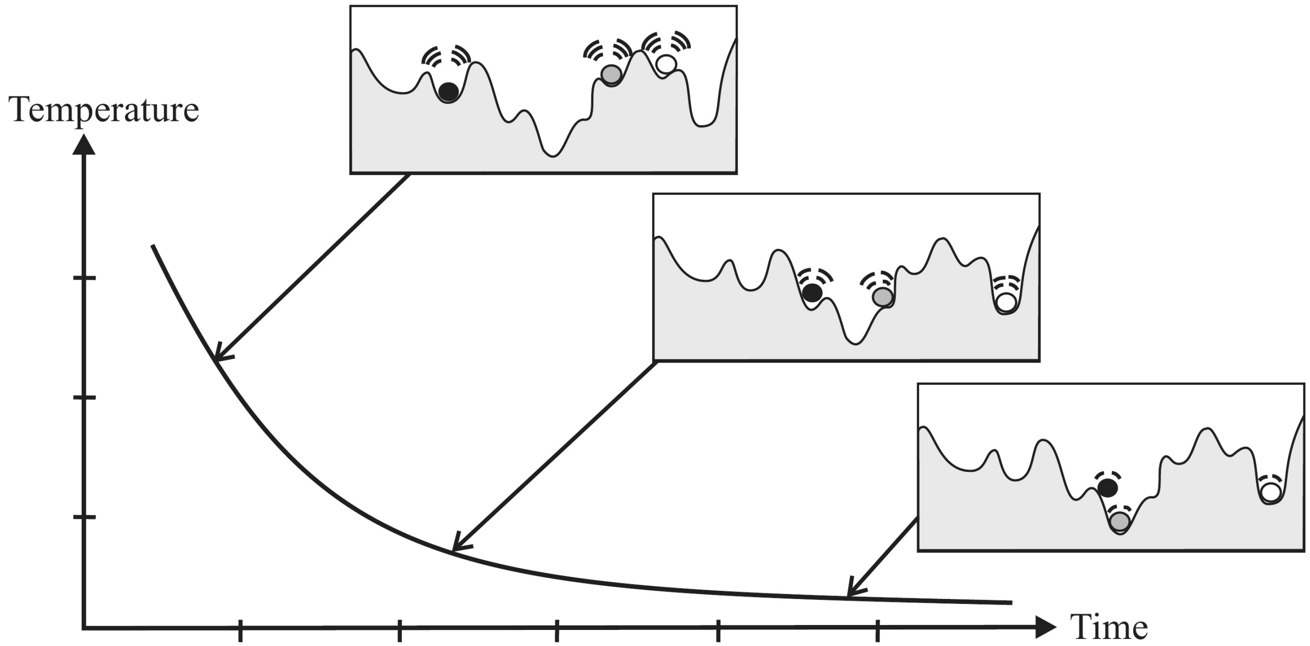
- Sometimes called greedy local search
- Hill-climbing can get stuck for several reasons
  - Local maxima
    - A peak higher than each of its neighbors but lower than the global maximum
  - Ridges
    - A sequence of local maxima that is difficult for greedy algorithms to navigate
  - Plateau
    - Flat area of the state-space landscape
      - Local maximum from which no uphill exists
      - Shoulder from which progress is possible
- In each case, the algorithm reaches a point at which no progress is being made

Simulated Annealing

- Hill-climbing is always vulnerable to getting stuck in a local maximum
  - At the other extreme, a pure random walk will eventually reach the global maximum
    - However, extremely inefficient
- Simulated annealing combines both worlds for yielding both efficiency and completeness
  - Annealing is the process to harden metals and glass by heating them to a high temperature
    - Then, gradually cooling the material allows it to reach a low-energy crystalline state

Simulated Annealing

- To understand simulated annealing let's view the problem as a gradient descent (that is, minimizing the cost)



Simulated Annealing Algorithm

- Pick a random move
  - If the move leads to an improvement, accept it
  - Otherwise, the move is accepted with some probability $p < 1$
    - The probability decreases exponentially according to the *badness* of a move

Idea: escape local maxima by allowing some "bad" moves but gradually decrease their size and frequency

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  current ← problem.INITIAL
  for t = 1 to ∞ do
     T ← schedule(t)
     if T = 0 then return current
     next ← a randomly selected successor of current
     ΔE ← VALUE(current) – VALUE(next)
     if ΔE > 0 then current ← next
     else current ← next only with probability e^(ΔE/T)
```

Figure 4.5 The simulated annealing algorithm, a version of stochastic hill climbing where some downhill moves are allowed. The *schedule* input determines the value of the "temperature" T as a function of time.

The probability decreases exponentially with the amount ΔE by which the evaluation is worsened.
The probability also decreases as the "temperature" T goes down: "bad" moves are more likely to be allowed at the start when T is high, and they become more unlikely as T decreases.

Properties of Simulated Annealing

- At fixed temperature T, state occupation probability reaches Boltzmann distribution

$$p(x) = ae^{\frac{E(x)}{kT}}$$

- T decreased slowly enough ⟹ always reach the best state x^* because

$$e^{\frac{E(x*)}{kT}} / \, e^{\frac{E(x)}{kT}} = e^{\frac{E(x*)-E(x)}{kT}}$$ -> 1 for small T

Local Beam Search

- The local beam search algorithm keeps track of k states rather than just one
  - Randomly generates k states
  - At each step, all the successors of all k are generated
    - If anyone is a goal, stop
    - Otherwise, select the k best successors from the complete list and repeat
- A local beam search with k states might seem as running parallel k random restarts instead of in sequence
  - However, in a random-restart search, each search process runs independently of the others, whereas, in a local beam search, useful information is passed among the parallel search threads
  - The algorithm quickly leaves unfruitful searches and moves its resources to where the most progress is being made
- A variant called stochastic beam search chooses successors with probability proportional to the successor's value to increase diversity

Evolutionary Algorithms

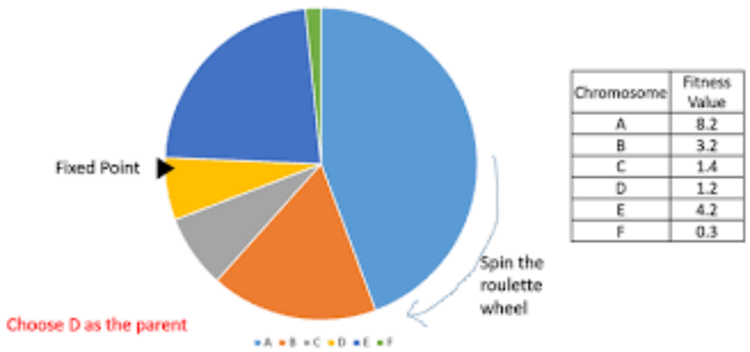
- Motivated by the metaphor of natural selection in biology
  - It is created a population of individuals (the state, that is, the solutions)
  - The fittest individuals (highest value) produce offspring (successor states)
    - This process is called recombination
    - The offspring after recombination form the next generation population
- Several variants of this evolutionary scheme exist
  - Genetic algorithms
  - Evolutionary strategy
  - Genetic programming

Genetic Algorithms (GAs)

- The population has a fixed size (number of individuals)
- Each individual, called a chromosome, is represented by a string over a finite alphabet (usually, a binary string)
- Each chromosome has a fitness value determining its goodness
- The population evolves through several generations
  - In each generation, the chromosomes are applied to three genetic operators
    - Selection
    - Crossover
    - Mutation

GA Operators: Selection

- Selects the chromosomes who will become parents of the next generation
  - The individuals are chosen with a probability proportional to their fitness value
    - This basic scheme is called roulette-wheel selection



Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

GA Operators: Crossover

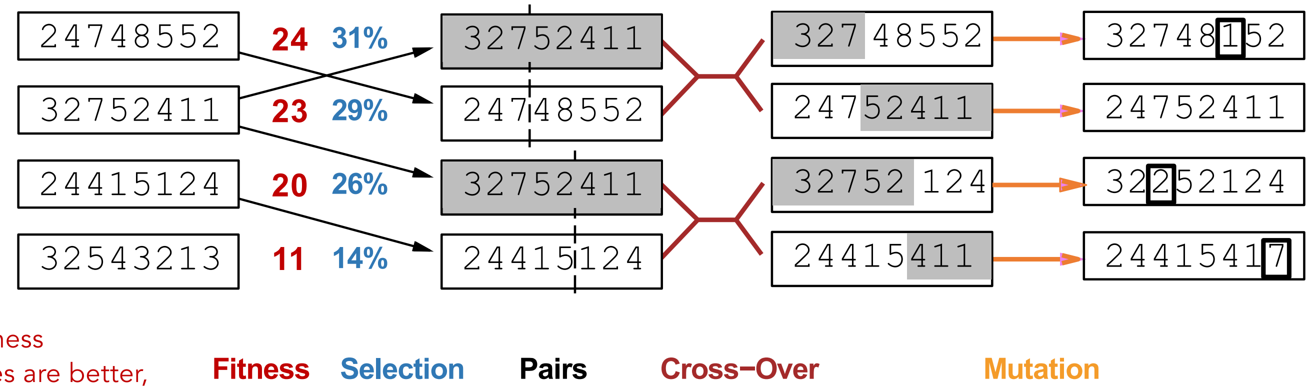
- Crossover is the operator for recombination
  - Once selected a pair of parents, it is randomly selected a crossover point where each parent string is split
  - The split substrings of one parent are recombined with the ones of the other parent to recombine and form the children (that, is the chromosome of the next generation)
    - one with the first part of parent 1 and the second part of parent 2
    - the other with the second part of parent 1 and the first part of parent 2

GA Operators: Mutation

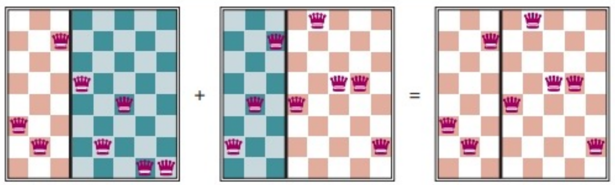
- Mutation randomly changes a symbol in the chromosome representation with a given probability
  - Once the offspring are generated, every bit in its representation is flipped with probability equal to a mutation rate
- Eventually, the next generation is formed
  - It could be just the newly formed offspring or
  - A few top-scoring parents from the previous generation are hold
    - Elitism
      - Guarantees that the overall fitness will never decrease over time

The Genetic Algorithm

- A GA with chromosomes representing 8-queens states
  - The initial population is ranked by a fitness function
  - The parents are then chosen for reproduction
  - The offspring are generated
  - Mutation possibly arises



Each state is rated by the fitness function: Higher fitness values are better, so we use the number of nonattacking pairs of queens, which has a value of 8×7/2 = 28 for a solution



How Gas Works

- Schema
  - a substring in which some of the positions can be left unspecified
    - the schema 246***** describes all 8-queens states in which the first three queens are in positions 2, 4, and 6, respectively
    - Strings that match the schema (such as 24613578) are called instances of the schema
  - It can be shown that if the average fitness of the instances of a schema is above the mean, then the number of instances of the schema will grow over time

GA Implementation

```
function GENETIC-ALGORITHM(population, fitness) returns an individual
  repeat
      weights ← WEIGHTED-BY(population, fitness)
      population2 ← empty list
      for i = 1 to SIZE(population) do
          parent1, parent2 ← WEIGHTED-RANDOM-CHOICES(population, weights, 2)
          child ← REPRODUCE(parent1, parent2)
          if (small random probability) then child ← MUTATE(child)
          add child to population2
      population ← population2
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to fitness

function REPRODUCE(parent1, parent2) returns an individual
  n ← LENGTH(parent1)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(parent1, 1, c), SUBSTRING(parent2, c + 1, n))
```

Figure 4.8 A genetic algorithm. Within the function, *population* is an ordered list of individuals, *weights* is a list of corresponding fitness values for each individual, and *fitness* is a function to compute these values.

Local Search in Continuous Spaces

- When the environment is continuous the branching factor is infinite, so the algorithms described so far are unsuitable
- Suppose we want to site three airports in Romania:
  - 6-D state space defined by (x1, y2), (x2, y2), (x3, y3)
  - objective function f (x1, y2, x2, y2, x3, y3) = sum of squared distances from each city to the nearest airport
  - This equation is correct for the state x and states in the local neighborhood of x
  - However, it is not correct globally; if we stray too far from x then the set of closest cities for that airport changes, and we need to recompute the nearest airports
- Discretization methods turn continuous space into discrete space, e.g., empirical gradient considers $\pm\delta$ change in each coordinate
- Gradient methods compute

$$\nabla f = \left(\frac{\partial f}{\partial x^1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y^3} \right)$$

to increase/reduce f by $\mathrm{x} \leftarrow \mathrm{x} + a\nabla f(\mathrm{x})$

Sometimes can solve for $\nabla f(x) = 0$ exactly (e.g., with one city). Newton–Raphson (1664, 1690) iterates $x \leftarrow x - H_f^{-1}(x)\nabla f(x)$ to solve $\nabla f(x) = 0$, where $H_{ij} = \partial^2 f / \partial x_i \partial x_j$