Evolutionary Computing
Genetic algorithms

Michela Mulas

A quick recap

Last lecture we introduced... Evolutionary computing

Darwin pioneered the idea that biological organisms develop and adapt over long periods of time via **descent with modification**.

- For example, organism **parents**, each with their own genetic makeup, mate, and their children's genetic makeup is a mixture of their parents so that often in appearance, we see characteristics of both parents.
- Sometimes, there are molecular **mutations** where an abnormal gene arises, which then affects the formation of the child.
- Both the mating and the mutations result in children growing up to be more or less fit to survive and mate in the environment that they live in.
- Children who are more fit tend to have more offspring, while children who are less fit often do not get the chance to mate, or have fewer offspring.

There is then a **natural selection** that proceeds gradually over time so that populations evolve to be more fit for their environment.

A quick recap

Last lecture we introduced... Evolutionary computing

Evolutionary computing refers to computer simulations that incorporate ideas from Darwin's theory on natural selection, and Lamarks work in genetics on inheritance, and try to simulate natural evolution of biological systems.

- **Survival of the fittest** refers to fitness in terms of reproductive success.
- Natural selection is the process where organisms with higher reproductive success generate offspring and propagate their DNA through time.
- Less fit individuals do not have offspring (or have fewer of them) and hence can become extinct over time.

A quick recap

Last lecture we introduced... Evolutionary computing

Evolutionary computing refers to computer simulations that incorporate ideas from Darwin's theory on natural selection, and Lamarks work in genetics on inheritance, and try to simulate natural evolution of biological systems.

- From an engineering perspective, the evolutionary computing can be seen as optimization techniques that evaluate more than one area of the search space and can discover more than one solution to a problem.
- Sometimes it is defined as a type of **stochastic direct search method**.
- It provides a stochastic optimization method where, **if it gets stuck at a local optimum**, it tries, via multiple search points, to simultaneously find other parts of the search space and **jump out of the local optimum to a global one** that represents the highest fitness individuals.

Today's goal

Today, we going to detail...

Genetic algorithms

- Basic concepts of optimisation
- A simple genetic algorithm example
- Solve an optimization problem in R

Reading list

A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing, Springer, 2007

A.P. Engelbrecht *Computational Intelligence – An Introduction*, Wiley 2007.

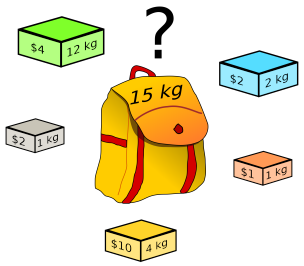
Optimisation problems

The knapsack problem

The knapsack problem is a generalisation of many industrial problems.

Given a set of n of items, each of which has a value v_i attached to it and some weight w_i. How do we select a subset of those items that maximises the value whilst keeping the summed cost within some capacity C_{max}?

- Find a combination of objects to put inside the knapsack.
- Start with an empty knapsack.
- Put objects in and take objects from the knapsack.
- Maximize the value: $\max \sum_i \frac{v_i}{w_i}$.



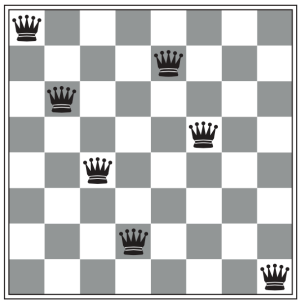
Optimisation problems

The eight queens problem

The **8 queens problem** aims at placing eight queens on a chessboard such that no queen attacks any other (A queen attacks any piece in the same row, column or diagonal).

Incremental formulation

- States: Any arrangement of 0 to 8 queens on the board is a state.
- Initial state: No queens on the board.
- Actions: Add a queen to any empty square.
- Transition model: Returns the board with a queen added to the specified square.
- Goal test: 8 queens are on the board, none attacked.
- There are $64 \cdot 63 \cdots 57 \approx 1.8 \times 10^{14}$ possible sequences to investigate.



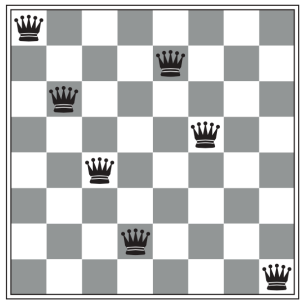
Optimisation problems

The eight queens problem

The **8 queens problem** aims at placing eight queens on a chessboard such that no queen attacks any other (A queen attacks any piece in the same row, column or diagonal).

Incremental formulation

- States: Any arrangement of 0 to 8 queens on the board is a state.
- Initial state: No queens on the board.
- Actions: Add a queen to any empty square.
- Transition model: Returns the board with a queen added to the specified square.
- Goal test: 8 queens are on the board, none attacked.
- There are $64 \cdot 63 \cdots 57 \approx 1.8 \times 10^{14}$ possible sequences to investigate.

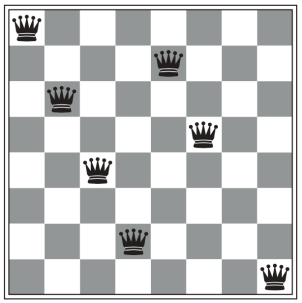


Optimisation problems
The eight queens problem

The **8 queens problem** aims at placing eight queens on a chessboard such that no queen attacks any other (A queen attacks any piece in the same row, column or diagonal).

Complete-state formulation

- States: All possible arrangements of n queens ($0 \leq n \leq 8$), one per column in the leftmost n columns, with no queen attacking another.
- Actions: Add a queen to any square in the leftmost empty column such that it is not attacked by any other queen.
- The are *only* 2057 possible sequences to investigate.



Optimisation methods
Main classes

An optimisation algorithm searches for an optimum solution by iteratively transforming a current candidate solution into a new, hopefully better, solution.

Based on the problem characteristics, optimisation methods are grouped in the following classes:

- **Unconstrained methods**, used to optimize unconstrained problems.
- **Constrained methods**, used to find solutions in constrained search spaces.
- **Multi-objective optimisation methods** for problems with more than one objective to optimize.
- **Multi-solution methods** with the ability to locate more than one solution.
- **Dynamic methods** with the ability to locate and track changing optima.

Optimisation methods
Main classes

An optimisation algorithm searches for an optimum solution by iteratively transforming a current candidate solution into a new, hopefully better, solution.

Optimisation methods can be divided into two main classes, based on the type of solution that is located.

- **Local search algorithms** use only local information of the search space surrounding the current solution to produce a new solution.
  - ⇝ Since only local information is used, local search algorithms locate local optima (it may be a global minimum).
- A **global search algorithm** uses more information about the search space to locate a global optimum.
  - ⇝ It is said that global search algorithms explore the entire search space, while local search algorithms exploit neighbourhoods.

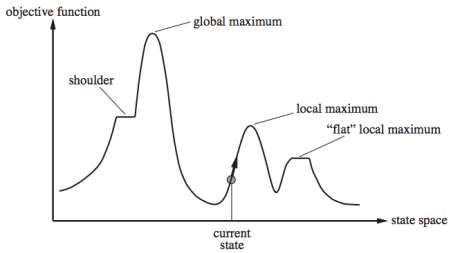
Optimisation methods
Local search algorithms

Local search algorithms are useful for solving pure optimisation problems, in which the aim is to find the best state according to an **objective function**.

Consider the state-space landscape, which has “location” (defined by the state) and “elevation” (defined by the value of objective function).

- If elevation is a cost: find the lowest valley - a **global minimum**.
- If elevation is an objective function: find the highest peak - **a global maximum**.

Local search algorithms explore this landscape.
Hill-climbing search modifies the current state to try to improve it.



Optimisation methods
Hill-climbing search

The hill-climbing search algorithm (**steepest-ascent** version) is simply a loop that continually moves in the direction of increasing value - that is, uphill. It terminates when it reaches a “peak” where no neighbour has a higher value.

- The data structure for the current node need only record the state and the value of the objective function.
- Hill climbing does not look ahead beyond the immediate neighbors of the current state.

Find the top of Mount Everest in a thick fog while suffering from amnesia

Optimisation methods
Hill-climbing search

The **hill-climbing often gets stuck** for the following reasons:

- **Local maxima**: a local maximum is a peak that is higher than each of its neighboring states but lower than the global maximum. Hill-climbing algorithms that reach the vicinity of a local maximum will be drawn upward toward the peak but will then be stuck with nowhere else to go.
- **Ridges** result in a sequence of local maxima that is very difficult for greedy algorithms to navigate.
- **Plateaux** is a flat area of the state-space landscape. It can be a flat local maximum, from which no uphill exit exists, or a shoulder, from which progress is possible. A hill-climbing search might get lost on the plateau.

Optimisation methods
Hill-climbing search

Many variants of hill climbing have been invented:

- **Stochastic hill climbing**: chooses at random from among the uphill moves; the probability of selection can vary with the steepness of the uphill move. This usually converges more slowly than steepest ascent, but in some state landscapes, it finds better solutions.
- **First-choice hill climbing** implements stochastic hill climbing by generating successors randomly until one is generated that is better than the current state. This is a good strategy when a state has many (e.g., thousands) of successors.
- **Random-restart hill**: conducts a series of hill-climbing searches from randomly generated initial states, until a goal is found.

The success of hill climbing depends very much on the shape of the state-space landscape: if there are few local maxima and plateaux, random-restart hill climbing will find a good solution very quickly.

Optimisation methods
Constrained optimisation

An optimisation problem is constrained if solutions must satisfy some hard constraints on the values of the variables.

- For example, we might constrain sites to be inside Romania and on dry land (rather than in the middle of lakes).

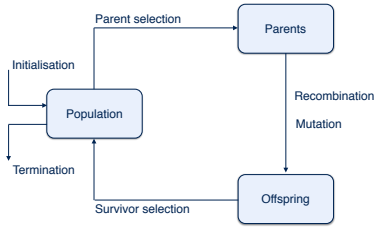
The difficulty of constrained optimisation problems depends on the nature of the constraints and the objective function.

- The best-known category is that of linear programming problems, in which constraints must be linear inequalities forming a convex set and the objective function is also linear.
- The time complexity of linear programming is polynomial in the number of variables.

Genetic algorithms

The genetic algorithm (GA) is the most widely known type of evolutionary algorithm.

- GA was conceived by Holland (1992) as a means of studying adaptive behaviour
- GAs traditionally have a fixed workflow.

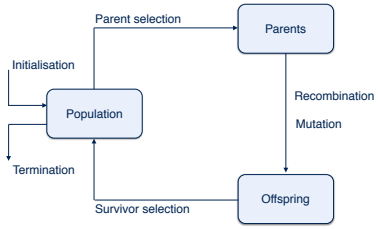


- Given a population of μ individuals, parent selection fills an intermediary population of μ, allowing duplicates.
- Then the intermediary population is shuffled to create random pairs.
- Crossover is applied to each consecutive pair with probability p_c.
- The children replace the parents immediately.

Genetic algorithms

The genetic algorithm (GA) is the most widely known type of evolutionary algorithm.

- GA was conceived by Holland (1992) as a means of studying adaptive behaviour
- GAs traditionally have a fixed workflow.

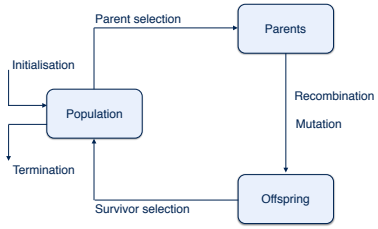


- The new intermediary population undergoes mutation individual by individual,
- Each of the l bits in an individual is modified by mutation with independent probability p_m.

Genetic algorithms

The genetic algorithm (GA) is the most widely known type of evolutionary algorithm.

- GA was conceived by Holland (1992) as a means of studying adaptive behaviour
- GAs traditionally have a fixed workflow.



- The resulting intermediary population forms the next generation replacing the previous one entirely.
- In this new generation there might be pieces, perhaps complete individuals, from the previous one that survived crossover and mutation without being modified, but the likelihood of this is rather low.

Genetic algorithms

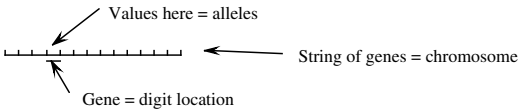
Representing the population of individuals implies...

- The **fitness function**: it measures the fitness of an individual to survive, mate, and produce offspring in a population of individuals for a given environment.
- The genetic algorithm will seek to maximize the fitness function $J(\theta)$ by selecting the individuals that we represent with θ.
- To represent the genetic algorithm in a computer, we make θ a string. It represents a **chromosome** in a biological system.

Genetic algorithms

Representing the population of individuals implies...

- A **chromosome is a string of genes** that can take on different **alleles**.
- In a computer, we often use number systems to encode alleles.
- We adopt the convention that a gene is a **digit location** that can take on different values from a number system (i.e., different types of alleles).



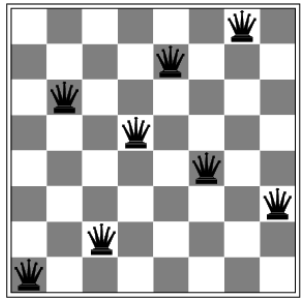
- In a base-2 number system, alleles come from the set $\{0, 1\}$
- In a base-10 number system, alleles come from the set {0,1, 2, 3, 4, 5, 6, 7, 8, 9}.
- A binary chromosome has zeros or ones in its gene locations.

Genetic algorithms
The eight queens problem

Consider again the **8-queens problem**.

Representation of the population

- Each state, or individual, is represented as a string over a finite alphabet-most commonly, a string of 0s and 1s.
- We assume that each queen has its own column.
- We represent the state with a row where the queen is in each column (digits 1 to 8).
- In the figure, the state is represented as `16257483`

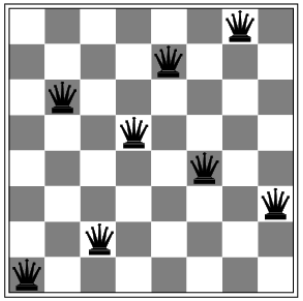


Genetic algorithms
The eight queens problem

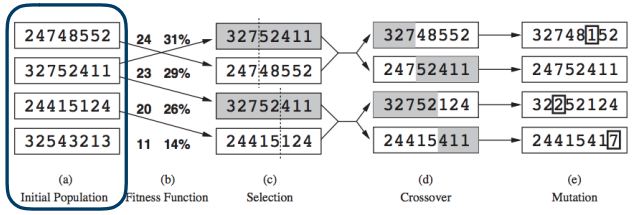
Consider again the **8-queens problem**.

Fitness function

- We use the number of non-attacking pairs of queens.
- There are 28 pairs of different queens (smaller columns first).
- The solution has fitness 28.
- The fitness function is $28 - h$, where h is the number of attaching pairs.
- In the figure, The fitness of the state is 27 (queens in column 4 and 7 attack each other).

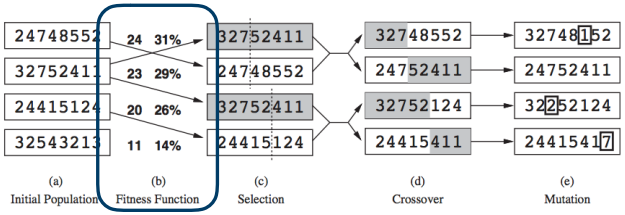


Genetic algorithms
The 8 queens problem



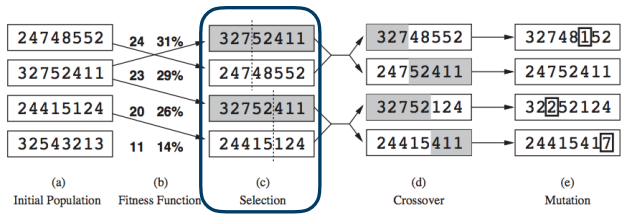
- GAs begin with a set of k randomly generated states, **the initial population**.
- Each state, or individual, is represented as a string over a finite alphabet.
  - ⇝ For example, an 8-queen state must specify the positions of 8 queens, each in a column of 8 squares.
  - ⇝ We chose to represent the state as 8 digits (from 1 to 8).

Genetic algorithms
The 8 queens problem



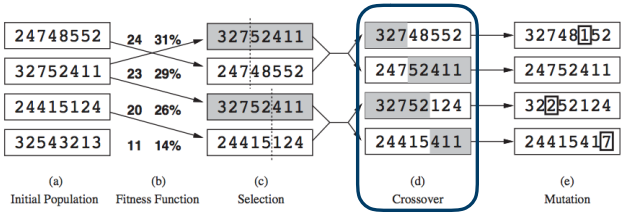
- Each state is rated by the objective function, **the fitness function**. It should return higher values for better states.
  - ⇝ The values of the four states are 24, 23, 20, and 11.
- The probability of being chosen for reproducing is directly proportional to the fitness score, and the percentages are shown next to the raw scores.

Genetic algorithms
The 8 queens problem



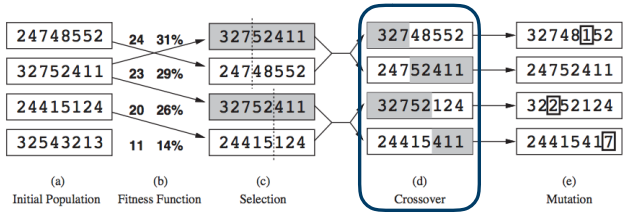
- Two pairs are selected at random for **reproduction**, according to the probabilities.
  - ⇝ Note that one individual is selected twice and one not at all.
- For each pair to be mated, a **crossover point** is chosen randomly from the positions in the string.
  - ⇝ Here, the crossover points are after the 3rd digit in the 1st pair and after the 5th digit in the 2nd pair.

Genetic algorithms
The 8 queens problem



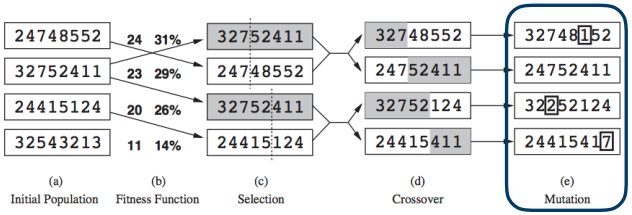
- The offspring themselves are created by crossing over the parent strings at the crossover point.
  - ⇝ The first child of the first pair gets the first three digits from the first parent and the remaining digits from the second parent.
  - ⇝ The second child gets the first three digits from the second parent and the rest from the first parent.

Genetic algorithms
The 8 queens problem



- The example shows that when two parent states are quite different, the crossover operation can produce a state that is a long way from either parent state.
- Often, the population is quite diverse early on in the process:
  - ⇝ Crossover frequently takes large steps in the state space early in the search process and smaller steps later on when most individuals are quite similar.

Genetic algorithms
The 8 queens problem



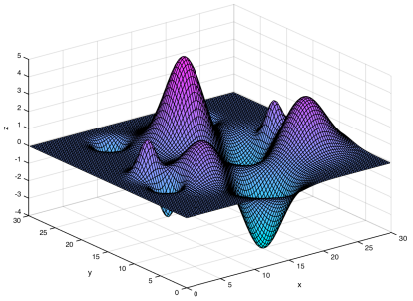
- Each location is subject to **random mutation** with a small independent probability.
- One digit was mutated in the first, third, and fourth offspring.
  - ⇝ In the 8-queens problem, this corresponds to choosing a queen at random and moving it to a random square in its column.

Genetic algorithms in R

In engineering design problems, there are many times when it is useful to solve some sort of optimization problem, since we often try to produce the “best” designs within a wide range of constraints (which include, e.g., cost).

In practical engineering problems, such optimization problems can be very difficult and at times it can be useful to turn to the genetic algorithm.

Suppose, we want to find the maximum of the function shown in figure using a genetic algorithm.



Genetic algorithms in R

In R, we can use genetic algorithms in different packages:

- `gafit`, `galts` and `mcga` offer some optimization routines based on GAs.
- `rgenoud` combines evolutionary algorithm methods with a derivative-based (quasi-Newton) method to solve optimization problems.
- `genalg` attempts to provide a genetic algorithm framework for both binary and floating points problems, but it is limited in scope and flexibility.

Genetic algorithms in R

In R, we can use genetic algorithms in different packages:

- `DEoptim` implements the differential evolution algorithm for global optimization of a real-valued function.
- `GA` gives a collection of general purpose functions that provide a flexible set of tools for applying a wide range of genetic algorithm methods.

Journal of Statistical Software
April 2013, Volume 53, Issue 4. http://www.jstatsoft.org/

GA: A Package for Genetic Algorithms in R

Luca Scrucca
Università degli Studi di Perugia