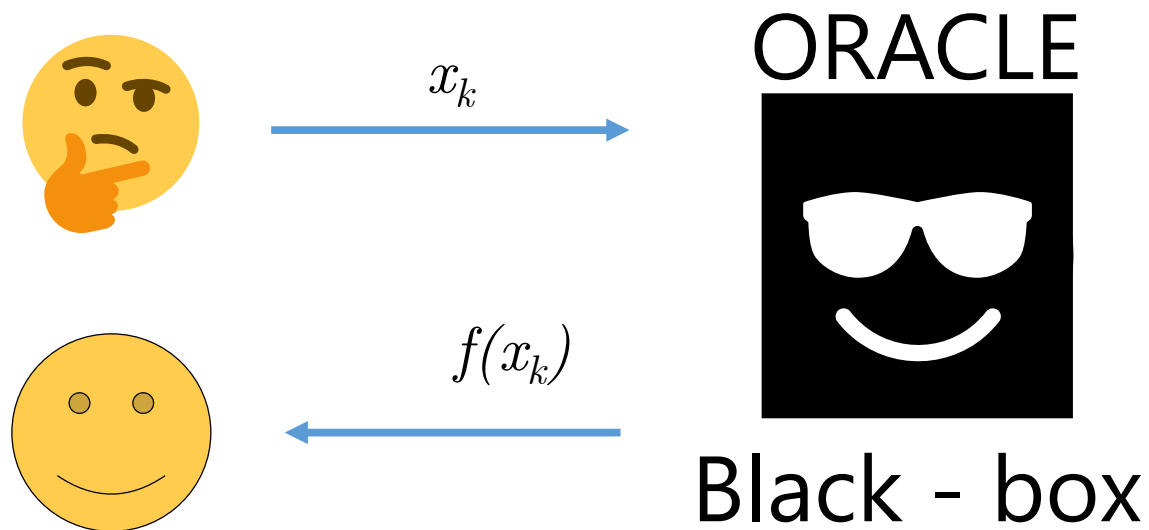


# Zero order methods

---



Now we have only zero order information from the oracle. Typical speed of convergence of these methods is sublinear. A lot of methods are referred both to zero order methods and global optimization.

## Code

---

- Global optimization illustration - [Open in Colab](#)
- Nevergrad library - [Open in Colab](#)

## Simulated annealing

---

### Problem

We need to optimize the global optimum of a given function on some space using only the values of the function in some points on the space.

$$\min_{x \in X} F(x) = F(x^*)$$

Simulated Annealing is a probabilistic technique for approximating the global optimum of a given function.

### Algorithm

The name and inspiration come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. Both are attributes of the material that depend on its thermodynamic free energy. Heating and cooling the material affects both the temperature and the thermodynamic free energy. The simulation of annealing can be used to find an approximation of a global minimum for a function with many variables.

## Steps of the Algorithm

**Step 1** Let  $k = 0$  - current iteration,  $T = T_k$  - initial temperature.

**Step 2** Let  $x_k \in X$  - some random point from our space

**Step 3** Let decrease the temperature by following rule  $T_{k+1} = \alpha T_k$  where  $0 < \alpha < 1$  - some constant that often is closer to 1

**Step 4** Let  $x_{k+1} = g(x_k)$  - the next point which was obtained from previous one by some random rule. It is usually assumed that this rule works so that each subsequent approximation should not differ very much.

**Step 5** Calculate  $\Delta E = E(x_{k+1}) - E(x_k)$ , where  $E(x)$  - the function that determines the energy of the system at this point. It is supposed that energy has the minimum in desired value  $x^*$ .

**Step 6** If  $\Delta E < 0$  then the approximation found is better than it was. So accept  $x_{k+1}$  as new started point at the next step and go to the step **Step 3**

**Step 7** If  $\Delta E \geq 0$ , then we accept  $x_{k+1}$  with the probability of  $P(\Delta E) = \exp^{-\Delta E/T_k}$ . If we don't accept  $x_{k+1}$ , then we let  $k = k + 1$ . Go to the step **Step 3**

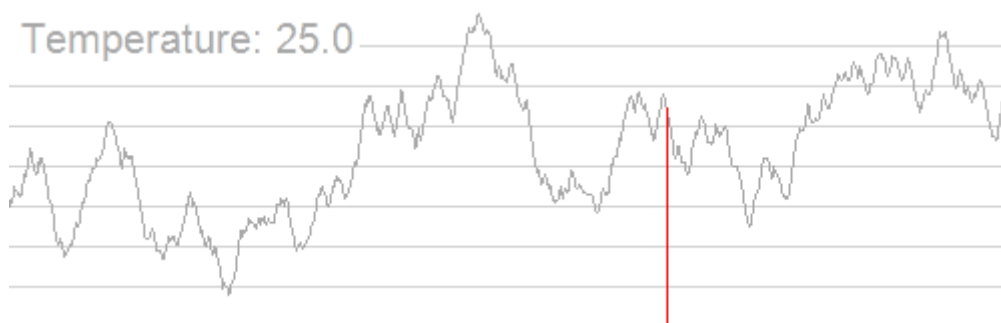
The algorithm can stop working according to various criteria, for example, achieving an optimal state or lowering the temperature below a predetermined level  $T_{min}$ .

## Convergence

As it mentioned in [Simulated annealing: a proof of convergence](#) the algorithm converges almost surely to a global maximum.









## Illustration

A gif from [Wikipedia](#):



## Example

In our example we solve the N queens puzzle - the problem of placing N chess queens on an N×N chessboard so that no two queens threaten each other.

## The Problem

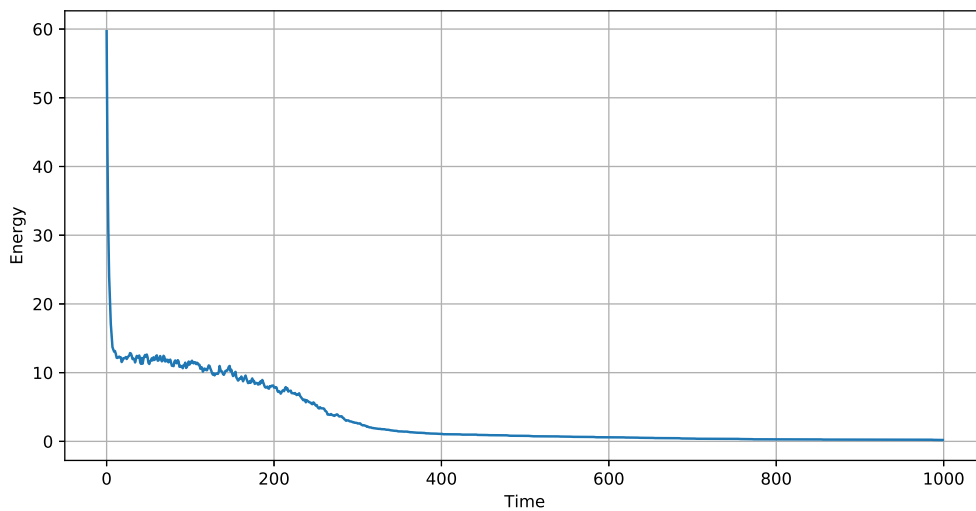
Let  $E(x)$  - the number of intersections, where  $x$  - the array of placement queens at the field (the number in array means the column, the index of the number means the row).

**The problem is** to find  $x^*$  where  $E(x^*) = \min_{x \in X} E(x)$  - the global minimum, that is predefined and equals to 0 (no two queens threaten each other).

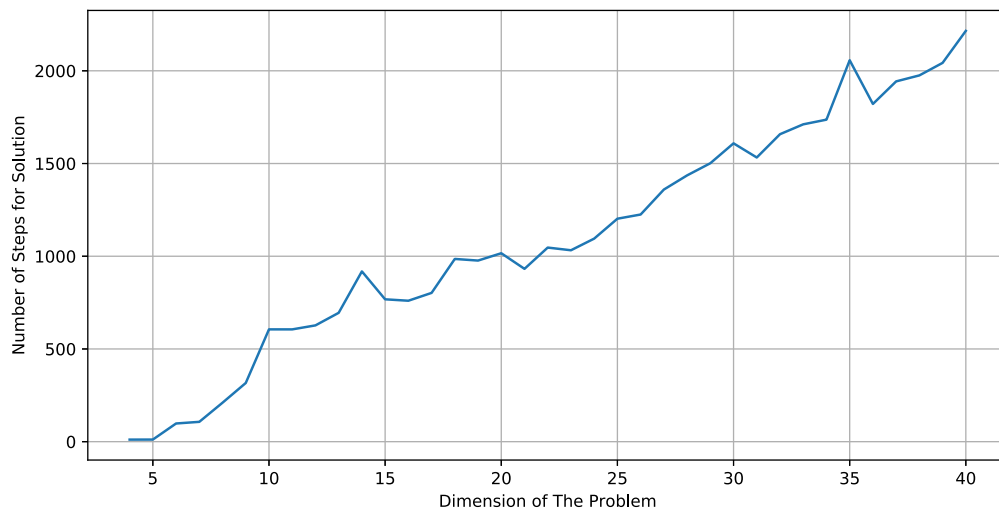
In this code  $x_0 = [0, 1, 2, \dots, N]$  that means all queens are placed at the board's diagonal. So at the beginning  $E = N(N - 1)$ , because every queen intersects others.

## Results

Results of applying this algorithm with  $\alpha = 0.95$  to the  $N$  queens puzzle for  $N = 10$  averaged by 100 runs are below:



Results of running the code for  $N$  from 4 to 40 and measuring the time it takes to find the solution averaged by 100 runs are below:



[Open in Colab](#)

# Genetic algorithm

## Problem

Suppose, we have  $N$  points in  $\mathbb{R}^d$  Euclidian space (for simplicity we'll consider and plot case with  $d = 2$ ). Let's imagine, that these points are nothing else but houses in some 2d village. Salesman should find the shortest way to go through the all houses only once.



That is, very simple formulation, however, implies  $NP$  - hard problem with the factorial growth of possible combinations. The goal is to minimize the following cumulative distance:

$$d = \sum_{i=1}^{N-1} \|x_{y(i+1)} - x_{y(i)}\|_2 \rightarrow \min_y,$$

where  $x_k$  is the  $k$ -th point from  $N$  and  $y$  stands for the  $N$ - dimensional vector of indices, which describes the order of path. Actually, the problem could be [formulated](#) as an LP problem, which is easier to solve.

## Genetic (evolution) algorithm

Our approach is based on the famous global optimization algorithm, known as evolution algorithm.

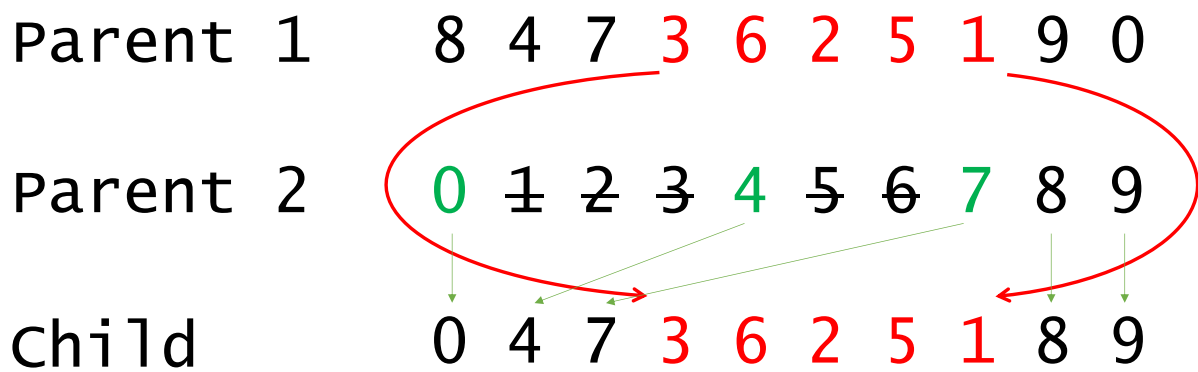
### Population and individuals

Firstly we need to generate the set of random solutions as an initialization. We will call a set of solutions  $\{y_k\}_{k=1}^n$  as *population*, while each solution is called *individual* (or creature).

Each creature contains integer numbers  $1, \dots, N$ , which indicates the order of bypassing all the houses. The creature, that reflects the shortest path length among the others will be used as an output of an algorithm at the current iteration (generation).

### Crossing procedure

Each iteration of the algorithm starts with the crossing (breed) procedure. Formally speaking, we should formulate the mapping, that takes two creature vectors as an input and returns its offspring, which inherits parents properties, while remaining consistent. We will use [ordered crossover](#) as such procedure.

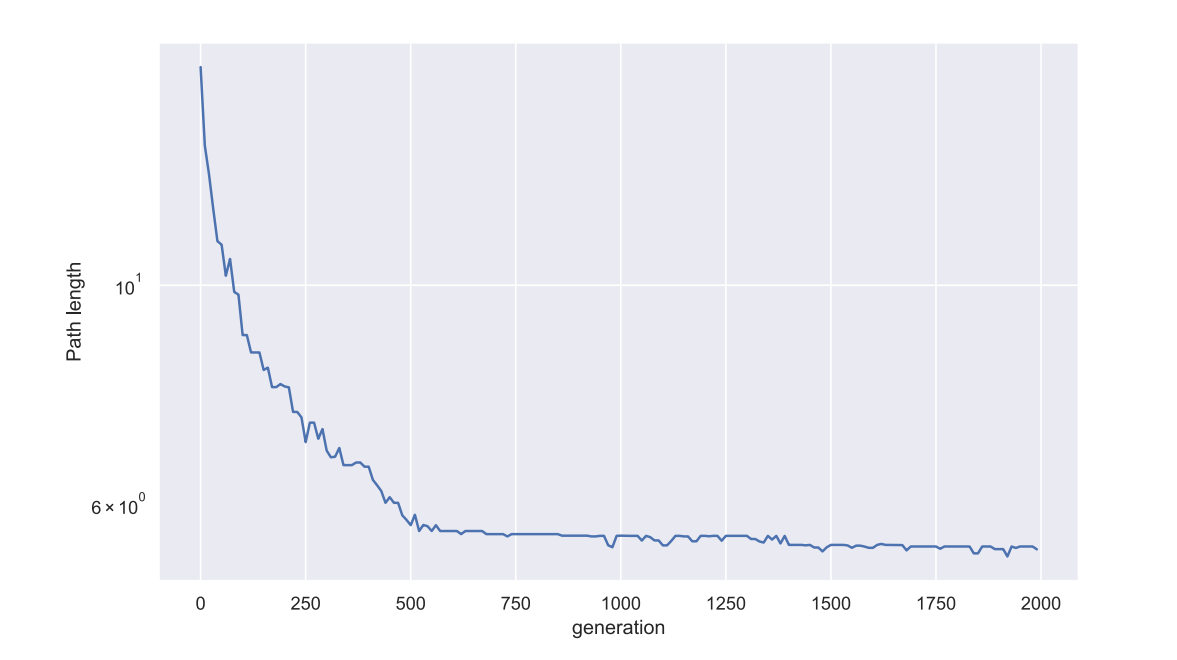
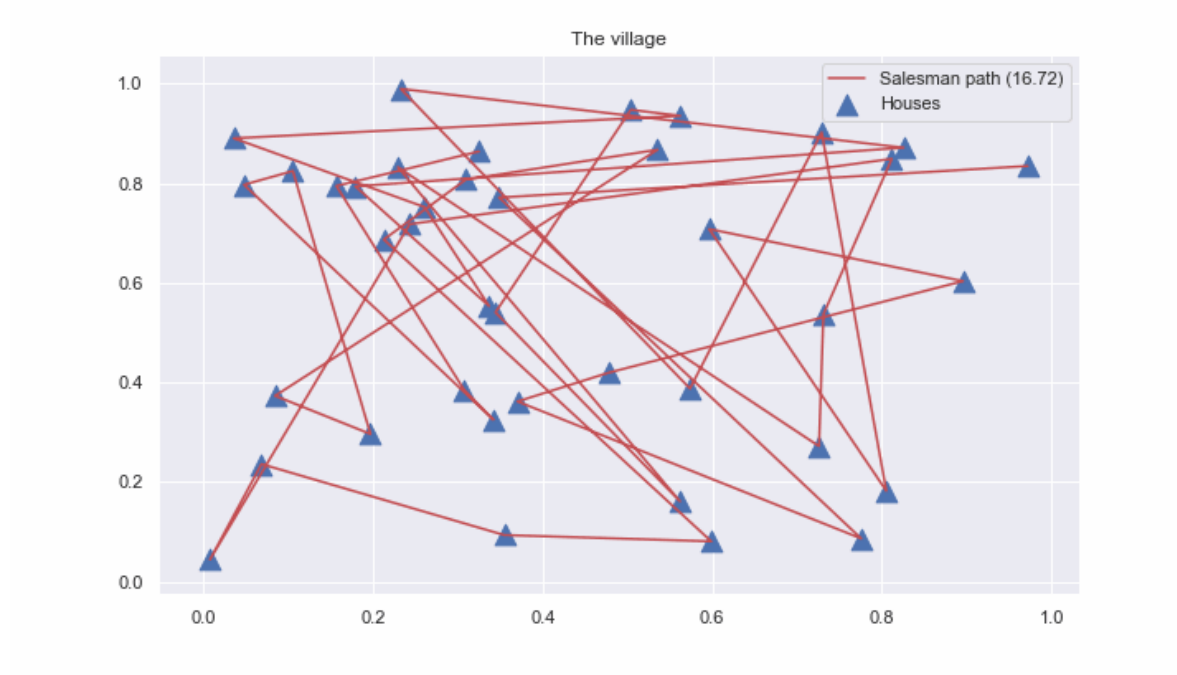


## Mutation

In order to give our algorithm some ability to escape local minima we provide it with mutation procedure. We simply swap some houses in an individual vector. To be more accurate, we define mutation rate (say, 0.05). On the one hand, the higher the rate, the less stable the population is, on the other, the smaller the rate, the more often algorithm gets stuck in the local minima. We choose  $\text{mutation rate} \cdot n$  individuals and in each case swap random  $\text{mutation rate} \cdot N$  digits.

## Selection

At the end of the iteration we have increased population (due to crossing results), than we just calculate total path distance to each individual and select top  $n$  of them.



In general, for any  $c > 0$ , where  $d$  is the number of dimensions in the Euclidean space, there is a polynomial-time algorithm that finds a tour of length at most  $(1 + \frac{1}{c})$  times the optimal for geometric instances of TSP in

$$\mathcal{O}\left(N(\log N)^{(\mathcal{O}(c\sqrt{d}))^{d-1}}\right)$$

## Code

 [Open in Colab](#)

## References

- [General information about genetic algorithms](#)
- [Wiki](#)

