Optimization

Màster de Fonaments de Ciència de Dades

Gerard Gómez

Lecture V. Stochastic methods: Genetic algorithms

Genetic algorithms

There is a large class of optimization problems for which no reasonably fast algorithms have been developed. Many of these problems are problems that arise frequently in applications.

For small spaces (low dimension), classical exhaustive methods usually suffice; for larger spaces special artificial intelligence techniques must be employed. Genetic Algorithms (GA) are among such techniques.

- Genetic algorithms were developed by John Holland (University of Michigan) in the early 1970's.
- They are stochastic search methods that mimic natural biological evolution.
- Genetic algorithms operate on a population of potencial solutions applying the principle of survival to generate improved estimations to a solution.
- At each generation, a new set of approximations is created by the process of selecting individuals according to their level of fitness and breeding them together, using genetic operators inspired by nature.
- This process leads to the evolution of better populations than previous populations.

Evolution programs

Genetic Algorithms belong to a wider class of methods that has been developed during the last thirty years.

They are procedures based on **principles of evolution and hereditary** and receive the generic name of **Evolution Programs** (EP).

Such methods:

- Maintain a **population** of potential solutions,
- Have some selection process based on **fitness of individuals**, and
- Use some **"genetic" operators.**

Some of such **Evolution Programs** (EP), are:

1. **Evolution Strategies** i.e., algorithms which imitate the principles of natural evolution for parameter optimization problems (Rechenberg, Schwefel).
2. Fogel's **Evolutionary Programming**, which is a technique for searching through a space of small finite-state machines.
3. Glover's **Scatter Search techniques** that maintain a population of reference points and generate offspring by weighted linear combinations.
4. Another type of evolution based systems are Holland's **Genetic Algorithms** (GA).

Evolution programs

- An **evolution program** is a probabilistic algorithm which maintains a population of individuals, $P(t) = (x_1^t, x_2^t, ..., x_n^t)$ for iteration t.
- Each x_i represents a **potential solution** to the problem, and is implemented as some data structure.
- Each solution x_i^t is evaluated to give some measure of its **"fitness"**.
- At each **iteration** $(t \rightarrow t+1)$ a **new population** is formed by selecting the more fit individuals (**select** step).
- Some members of the new population undergo transformations by means of **"genetic"** operators to form new solutions.
  - There are transformations m_i (**mutation** type), which create new individuals by a small change in a single individual, and
  - Higher order transformations cj (**crossover** type), which create new individuals by combining parts from several (two or more) individuals.
- After "some number" of generations the program converges - it is hoped that the best individual represents a near-optimum (reasonable) solution.

Structure of an evolution program

```
begin
    0 → t
    initialize P(t)
    evaluate P(t)
    while (not termination-condition) do
        t + 1 → t
        select P(t) from P(t − 1)
        alter P(t)
        evaluate P(t)
    end do
end
```

Genetic algorithms. What nature does

Consider a population of rabbits and foxes.

- At any given time there is a population of rabbits, some of them are faster and/or smarter than other rabbits.
- These faster and/or smarter rabbits are less likely to be eaten by foxes, and therefore more of them survive to do what rabbits do best: make more rabbits.
- Of course, some of the slower, dumber rabbits will survive just because they are lucky.
- The surviving population of rabbits starts breeding. The breeding results in a good mixture of rabbit genetic material: some slow rabbits breed with fast rabbits, some fast with fast, some smart rabbits with dumb rabbits, and so on.
- And on the top of that nature, every once in a while, mutates some of the rabbit genetic material.
- The resulting baby rabbits will (on average) be faster and smarter than these in the original population because more faster, smarter parents survived the foxes. (It is a good thing that the foxes are undergoing similar process - otherwise the rabbits might become too fast and smart for the foxes to catch any of them).

Genetic algorithms. Example

Assume we search for a graph which should satisfy some requirements; for instance: we search for the optimal topology of a communication network accordingly to some criteria: cost of sending messages, reliability, etc.

- Each individual in the evolution program represents a potential solution to the problem, i.e., each individual is or represents a graph.
- The initial population of graphs $P(0)$ (either generated randomly or created as a result of some heuristic process) is a starting point ($t = 0$) for the evolution program.
- The evaluation function usually is given and incorporates the problem requirements. The evaluation function returns the fitness of each graph, distinguishing between better and worse individuals.
- A few crossover operators can be considered which combine the structure of two (or more) graphs into one.
- Several mutation operators can be designed which would transform a single graph.
- Often such operators incorporate the problem-specific knowledge. For example, if the graph we look for is a connected tree, a possible mutation operator may delete an edge from the graph and add a new edge to connect two disjoint subgraphs.
- The other possibility would be to design a problem-independent mutation and incorporate this requirement into the evaluation function, penalizing graphs which are not trees.

Structure of a GA

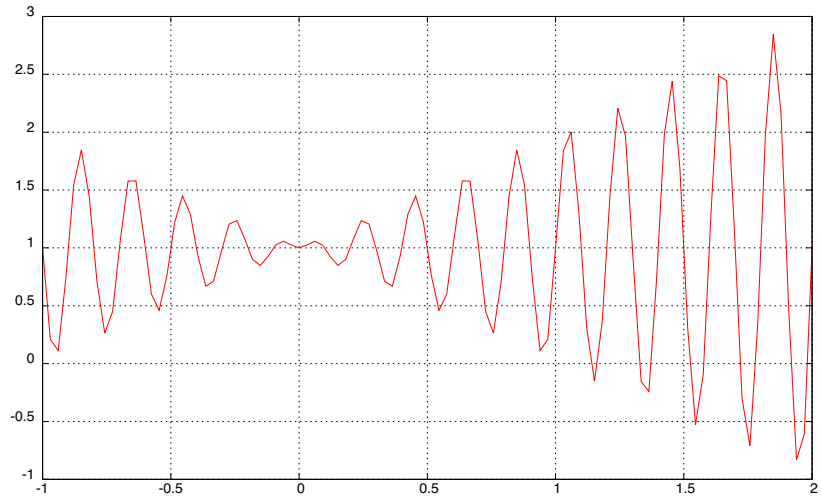
The structure of a simple genetic algorithm is the same as the structure of any evolution program.

- During iteration t, a genetic algorithm maintains a population of potential solutions (chromosomes, vectors), $P(t) = (x_1^t, ..., x_n^t)$.
- Each solution x_i^t is evaluated to give some measure of its "fitness".
- Then, a new population (iteration $t + 1$) is formed by selecting the more fit individuals.
- Some members of this new population undergo alterations by means of crossover and mutation, to form new solutions.
- Crossover combines the features of two parent chromosomes to form two similar offspring by swapping corresponding segments of the parents. For example, if the parents are represented by five-dimensional vectors $(a_1, b_1, c_1, d_1, e_1)$ and $(a_2, b_2, c_2, d_2, e_2)$, then crossing the chromosomes after the second gene would produce the offspring $(a_1, b_1, c_2, d_2, e_2)$ and $(a_2, b_2, c_1, d_1, e_1)$
- The intuition behind the applicability of the crossover operator is information exchange between different potential solutions.

Optimization of a function

Consider the following problem: find $x \in [-1, 2]$ that maximizes the function

$$f(x) = x \sin(10\,\pi\,x) + 1.$$



The extrema of f are the zeros of

$$f'(x) = \sin(10\,\pi\,x) + 10\,\pi\,x\cos(10\,\pi\,x) = 0 \quad \Leftrightarrow \quad \tan(10\,\pi\,x) = -10\,\pi\,x$$

They are

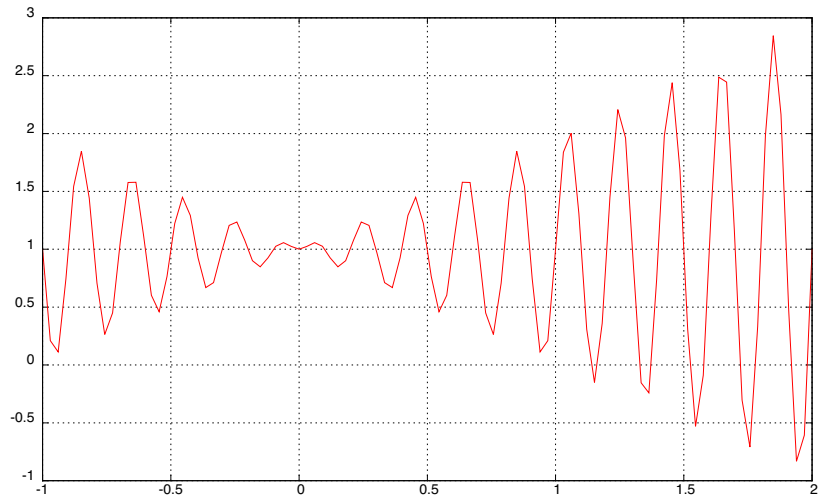
$$\begin{aligned} x_i &= \frac{2i+1}{20} - \epsilon_i, \quad \text{for} \quad i = -1, -2, \ldots \\ x_0 &= 0 \\ x_i &= \frac{2i-1}{20} + \epsilon_i, \quad \text{for} \quad i = 1, 2, \ldots \end{aligned}$$

where $\{\epsilon_i\}$ is a decreasing sequence of real numbers approaching zero.

Optimization of a function

The function f reaches its local maxima at x_i if i is an odd number and its local minima for x_i if i is even.

Since the domain of the problem is $[-1, 2]$, the function reaches its maximum for for $x_{19} = 37/20 + \epsilon_{19} \approx 1.85$ and the value of $f(x_{19})$ is approximately 2.28.



We wish to construct a genetic algorithm to solve the above problem, i.e., to maximize the function f.

Optimization of a function. Representation

- We use a binary vector as a chromosome to represent real values of the variable x.
- The length of the vector depends on the required precision. In this example, the required precision will be six places after the decimal point.
- The precision requirement implies that the range $[-1.0, 2.0]$ should be divided into, at least, $3 \times 1\,000\,000$ equal size ranges.
- Since
$$20\,971\,522^{21} < 3\,000\,000 < 2^{22} = 4\,194\,304,$$
this means that 22 bits are required as a binary vector (chromosome).
- The mapping from a binary string $(b_{21}\, b_{20} \ldots b_0)_2$ into a real number $x \in [-1.0, 2.0]$ is straightforward and is completed in two steps:
  - Convert the binary string $(b_{21}\, b_{20} \ldots b_0)_2$ from the base 2 to base 10:
$$(b_{21}\, b_{20} \ldots b_0)_2 = \sum_{i=0}^{21} b_i\, 2^i = x' \in [0, 2^{22} - 1]$$
  - Find a corresponding real number x in the range $[-1, 2]$:
$$\frac{x + 1.0}{3.0} = \frac{x'}{2^{22} - 1} \quad \Rightarrow \quad x = -1.0 + \frac{3.0}{2^{22} - 1} x'.$$

Optimization of a function. Representation

- For instance, if the chromosome is $(10001011101101010001 11)_2$, it represents the real number $x = 0.637196$, since

$$(10001011101101010001 11)_2 = 2288967$$

and

$$x = -1.0 + \frac{3}{4194303} 2288967 = 0.637196$$

- Of course, the chromosomes

$$(00000000000000000000)_2 \quad \text{and} \quad (11111111111111111111)_2$$

represent the boundaries of the domain, -1.0 and 2.0, respectively.

Optimization of a function. Initial population

The **initialization process** is very simple: we create a population of chromosomes, where each chromosome is a binary vector of 22 bits.

All 22 bits for each chromosome are **initialized randomly.**

Optimization of a function. Evaluation function

The **valuation function** *eval* for binary vectors v is the **fitness function** f:

$$eval(v) = f(x),$$

where the chromosome v represents the real value x.

The evaluation function plays the role of the environment, rating potential solutions in terms of their fitness.

For example, three chromosomes:

$$\begin{aligned} v_1 &= (10001011101101010001 11)_2 \\ v_2 &= (00000011100000000010000)_2 \\ v_3 &= (11100000001111111000101)_2 \end{aligned}$$

correspond to values $x_1 = 0.637197$, $x_2 = -0.958973$, and $x_3 = 1.627888$, respectively. Consequently, the evaluation function would rate them as follows:

$$\begin{aligned} eval(v_1) &= f(x_1) = 1.586345 \\ eval(v_2) &= f(x_2) = 0.078878 \\ eval(v_3) &= f(x_3) = 2.250650 \end{aligned}$$

Clearly, the chromosome v_3, is the best of the three chromosomes, since its evaluation returns the highest value.

Optimization of a function. Genetic operations

- During the alteration phase of the genetic algorithm we would use two classical genetic operators: **mutation** and **crossover**.
- Mutation **alters one or more genes** (positions in a chromosome) with a **probability** equal to the **mutation rate**.
- Usually the probability of mutation p_m is chosen to be very low (e.g., 0.01).
- Assume that the fifth gene from the v_3 chromosome was selected for a mutation. Since the fifth gene in this chromosome is 0, it would be flipped into 1. So the chromosome v_3 after this mutation would be

$$v_3 = (11101000001111111000101)_2$$

- This chromosome represents the value $x_3' = 1.721638$ and $f(x_3') = -0.082257$. This means that this particular mutation resulted in a significant decrease of the value of the chromosome v_3
- On the other hand, if the 10th gene was selected for mutation in the chromosome v_3, then

$$v_3 = (11100000011111111000101)_2$$

Then $x_3'' = 1.630818$ and $f(x_3'') = 2.343555$, and we get an improvement over the original value of $f(x_3) = 2.250650$.

Optimization of a function. Genetic operations

- Let us illustrate the **crossover** operator on chromosomes v_2 and v_3.
- Assume that the **crossover point** was **randomly** selected after the 5th gene:

$$\begin{aligned} v_2 &= (00000\,|\,0111000000010000)_2 \\ v_3 &= (11100\,|\,0000111111000101)_2 \end{aligned}$$

The two resulting offspring are

$$\begin{aligned} v_2' &= (00000\,|\,0000111111000101)_2 \\ v_3' &= (11100\,|\,0111000000010000)_2 \end{aligned}$$

These offspring evaluate to

$$\begin{aligned} f(v_2') &= f(-0.998113) = 0.940865 \\ f(v_3') &= f(1.666028) = 2.459245. \end{aligned}$$

Note that the second offspring has a better evaluation than both of its parents (0.078878 and 2.250650).

- Usually, the probability of crossover p_c is chosen to be fairly high (e.g., 0.7).

Optimization of a function. Experimental results

Consider the following simulation parameters for this problem:

- population size $= 50$,
- probability of crossover $p_c = 0.25$,
- probability of mutation $p_m = 0.01$.

The table provides the generation number (for 150 generations) for which there is an improvement in the evaluation function, together with the value of the function.

Generation number	Evaluation function
1	1.441942
6	2.250003
8	2.250283
9	2.250284
10	2.250363
12	2.328077
39	2.344251
40	2.345087
51	2.738930
99	2.849246
137	2.850217
145	2.850227

Optimization of a function. Experimental results

- The best chromosome, after 150 generations, was

$$v_{max} = (11110011010000100000101)$$

  which corresponds to a value $x_{max} = 1.850773$.

- As expected, $x_{max} = 1.85 + \epsilon$, and $f(x_{max}) = 2.850227$ is slightly larger than 2.85.

How do GA work

- Suppose we wish to maximize a function of k variables

$$f(x_1, ..., x_k) : \mathbb{R}^k \to \mathbb{R}$$

- Suppose further that each variable x_i can take values from a domain $D_i = [a_i, b_i] \subset \mathbb{R}$, and $f(x_1, ..., x_k) > 0$ for all $x_i \in D_i$ (if the original f takes negative values, we can add some positive constant C to it).

- We wish to optimize the function f with some required precision: suppose six decimal places for the variables' values is desirable.

How do GA work

- It is clear that to achieve such precision each domain $D_i = [a_i, b_i]$ should be cut into $(b_i - a_i) \times 10^6$ equal size ranges.
- Denote by m_i the smallest integer such that $(b_i - a_i) \times 10^6 \leq 2^{m_i} - 1$. Then, a representation having each variable x_i coded as a binary string of length m_i clearly satisfies the precision requirement.
- The following formula gives the transformation from binary to decimal:

$$x_i = a_i + \frac{b_i - a_i}{2^{m_i} - 1} decimal(binary\ string)_2.$$

- Now, each chromosome (as a potential solution) is represented by a binary string of length

$$m = \sum_{i=1}^{k} m_i,$$

the first m_1 bits map into a value frorn the range $[a_1, b_1]$, the next group of m_2 bits map into a value from the range $[a_2, b_2]$, and so on.

How do GA work

- To **initialize a population**, we can simply set some number of chromosomes randomly in a bitwise fashion.
- If we do have some knowledge about the distribution of potential optima, we may use such information in arranging the set of initial (potential) solutions.
- The rest of the algorithm is straightforward: in each generation we **evaluate each chromosome** (using the function f on the decoded sequences of variables), **select new population** with respect to the probability distribution based on fitness values, and **alter the chromosomes** in the new population by **mutation and crossover** operators.
- After some number of generations, when no further improvement is observed, the best chromosome represents an (possibly the global) optimal solution.
- Often we stop the algorithm after a fixed number of iterations depending on speed and resource criteria.

The selection process

The selection of a new population with respect to the probability distribution based on fitness values.

It can be done using a **roulette wheel with slots**, sized according to fitness is used. We construct such a roulette wheel as follows:

- Calculate the **fitness value** $eval(v_i)$ **for each chromosome** v_i, $i = 1, ...,$ population-size.
- Find the **total fitness** of the population

$$F = \sum_{i=1}^{pop.size} eval(v_i)$$

- Calculate the **probability of a selection** p_i for each chromosome v_i

$$p_i = \frac{eval(v_i)}{F}$$

- Calculate a **cumulative probability** q_i for each chromosome v_i

$$q_i = \sum_{j=1}^{i} p_j$$

The selection process. The roulette wheel with slots

The **selection process** is based on spinning the roulette wheel as many times as the population size ($pop - size$). Each time we select a single chromosome for a new population in the following way:

- Generate a random number r in the range $[0, 1]$
- If $r < q_1$ then select the first chromosome v_1.
- Otherwise select the $i - th$ chromosome v_i $(2 \leq i \leq pop - size)$ such that

$$q_{i-1} < r \leq q_i.$$

Obviously, some chromosomes would be selected more than once. This is in accordance with the "Schema Theorem": the best chromosomes get more copies, the average stay, and the worst die off.

Now we are ready to apply the recombination operator, **crossover**, to the individuals in the new population.

Other selection process. Tournament selection

Tournament selection

- In tournament selection, two chromosomes are chosen at randon and with replacement from the population.
- Let c be a constant chosen by the user to be between 0 and 1.
- A randon number r, $0 \leq r \leq 1$ is drawn, if $r < c$ the best chromosome is selected for parenthood.

Other selection process. Stochastic universal sampling

Stochastic universal sampling

- Stochastic universal sampling works by making a single spin of the roulette wheel.
- This provides a starting position and the first selected individual.
- The selection process then proceeds by advancing all the way around the wheel in equal sized steps, where the step size is determined by the number of individuals to be selected.
- Note that this does not mean that every candidate on the wheel will be selected.
- Some weak individuals will have very thin slices of the wheel and these might be stepped over completely depending on the random starting position.

The crossover

The **probability of crossover**, p_c, is one of the key parameters of a genetic algorithm.

This probability **gives the expected number of chromosomes which undergo the crossover** operation.

We proceed in the following way:

- For each chromosome in the new population, generate a random number r from the range $[0, 1]$
- If $r < p_c$ select given chromosome for crossover.
- Next we mate all the selected chromosomes randomly:
  - For each pair of selected chromosomes we generate a random integer number, pos, from the range $[1, ..., m-1]$ (m is the total length - number of bits - in a chromosome).
  - The number pos indicates the position of the crossing point. Two chromosomes

$$(b_1, b_2, ..., b_{pos}, b_{pos+1}, ..., b_m) \quad \text{and} \quad (c_1, c_2, ..., c_{pos}, c_{pos+1}, ..., c_m)$$

are replaced by a pair of their offspring

$$(b_1, b_2, ..., b_{pos}, c_{pos+1}, ..., c_m) \quad \text{and} \quad (c_1, c_2, ..., c_{pos}, b_{pos+1}, ..., b_m)$$

The mutation

- The next operator, **mutation**, is performed on a bit-by-bit basis.
- The probability of mutation p_m, gives us the expected number of mutated bits p_m.
- Every bit (in all chromosomes in the whole population) has an equal chance to undergo mutation, i.e., change from 0 to 1 or vice versa. So we proceed in the following way:
  - For each chromosome in the current (i.e., after crossover) population, and for each bit within the chromosome, generate a random number r in the range $[0, 1]$
  - If $r < p_m$, mutate the bit.

Following selection, crossover, and mutation, the new population is ready for its next evaluation. This evaluation is used to build the probability distribution (for the next selection process), i.e., for a construction of a roulette wheel with slots sized according to current fitness values.

The rest of the evolution is just cyclic repetition of the above steps.

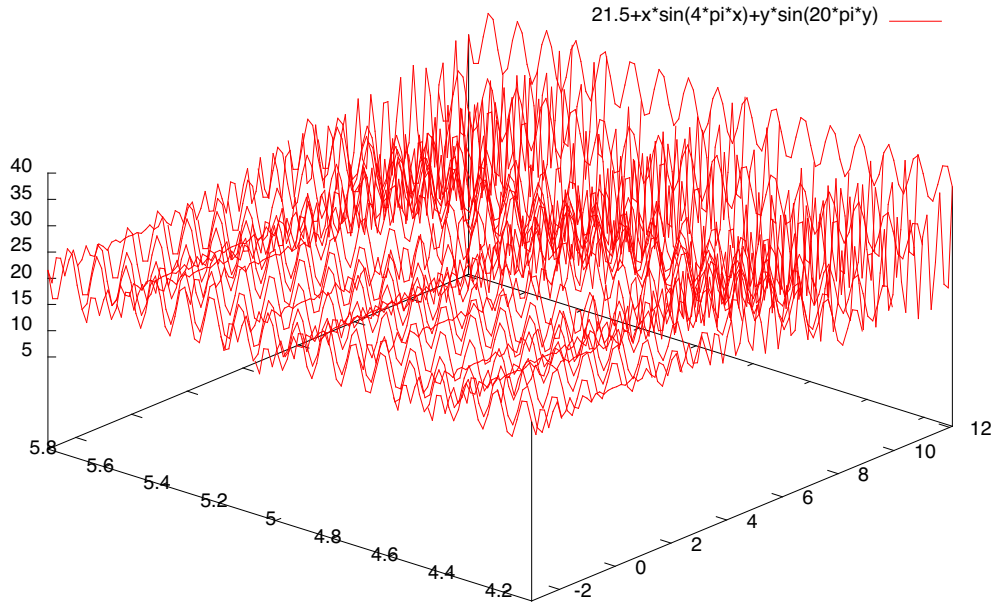
Example

We want to maximize the following function

$$f(x, y) = 21.5 + x \sin(4\,\pi\, x) + y \sin(20\,\pi\, y),$$

with

$$(x, y) \in [-3.0, 12.11] \times [4.5, 5.8]$$



We will use a GA with a population size $pop - size = 20$, and as probabilities of the genetic operators of crossover $p_c = 0.25$ and mutation $p_m = 0.01$.

Example (cont.)

- Assume that the required precision is four decimal places for each variable.
- The domain of variable x llas length 15.1. The precision requirement implies that the x-range $[-3.0, 12.11]$ should be divided into at least $15.1 \times 10\,000$ equal size ranges. This means that 18 bits are required as the first part of the chromosome:

$$2^{17} < 151\,000 \leq 2^{18}$$

- The domain of variable y has length 1.7. The precision requirement implies that the range $[4.1, 5.8]$ should be divided into at least $1.7 \times 10\,000$ equal size ranges. This means that 15 bits are required as the second part of the chromosome:

$$2^{14} < 17\,000 \leq 2^{15}$$

- The total length of a chromosome (solution vector) is then $m = 18 + 15 = 33$ bits; the first 18 bits code x and remaining 15 bits $(19 - 33)$ code y.

Example (cont.)

- Let us consider an example chromosome

$$(010001001011010000111110010100010)$$

- The first 18 bits

$$010001001011010000$$

represent

$$x = -3 + \frac{12.1 - (-3.0)}{2^{18} - 1} decimal(010001001011010000)_2 = 1.052426$$

The next 15 bits

$$111110010100010$$

represent

$$y = 4.1 + \frac{5.8 - 4.1}{2^{15} - 1} decimal(111110010100010)_2 = 5.755330$$

So the chromosome corresponds to $(x, y) = (1.052426, 5.755330)$.

- The fitness value for this chromosome is

$$f(1.052426, 5.755330) = 20.252640.$$

Example (cont.)

Assume that after the initialization process we get the following population:

$$
\begin{aligned}
v_1 &= (10011010000000111111010011011111)\\
v_2 &= (11100010010011011100101010011010)\\
v_3 &= (00001000001100100000101011011101)\\
v_4 &= (10001100010110100111100000111010)\\
v_5 &= (00011101100101001101011111000101)\\
v_6 &= (00010100001001010100101011111011)\\
v_7 &= (00100010000011010111101101111011)\\
v_8 &= (10000110000111010001011010110011 1)\\
v_9 &= (01000000101110100010110101100111)\\
v_{10} &= (00000111100011000001101000011101 1)\\
v_{11} &= (01100111111011010110000110111 1000)\\
v_{12} &= (11010001011110110100010101000000 0)\\
v_{13} &= (11101111101000100011000000100011 0)\\
v_{14} &= (00001001100000101010011110010100 1)\\
v_{15} &= (11101110110111000010001111101111 0)\\
v_{16} &= (11001111000001111110000110100101 1)\\
v_{17} &= (01101011111100111101000110111110 1)\\
v_{18} &= (01110100000000111101001111010110 1)\\
v_{19} &= (00010101001111111111000011000110 0)\\
v_{20} &= (10111001011001111001100010111111 0)
\end{aligned}
$$

Example (cont.)

During the evaluation phase we decode each chromosome and calculate the fitness function values from (x, y) values just decoded. We get:

$$
\begin{array}{ll}
eval(v_1) = 26.019600 & eval(v_2) = 7.580015 \\
eval(v_3) = 19.526329 & eval(v_4) = 17.406725 \\
eval(v_5) = 25.341160 & eval(v_6) = 18.100417 \\
eval(v_7) = 16.020812 & eval(v_8) = 17.959701 \\
eval(v_9) = 16.127799 & eval(v_{10}) = 21.278435 \\
eval(v_{11}) = 23.410669 & eval(v_{12}) = 15.011619 \\
eval(v_{13}) = 27.316702 & eval(v_{14}) = 19.876294 \\
eval(v_{15}) = 30.060205 & eval(v_{16}) = 23.867227 \\
eval(v_{17}) = 13.696165 & eval(v_{18}) = 15.414128 \\
eval(v_{19}) = 20.095903 & eval(v_{20}) = 13.666916
\end{array}
$$

It is clear, that the chromosome v_{15} is the best one, and the chromosome v_2 the worst.

Next, construct a roulette wheel for the selection process.

The total fitness of t.he population is

$$F = \sum_{i=1}^{20} eval(v_i) = 387.776822$$

Example (cont.)

The probability of a selection p_i for each chromosome is:

$$
\begin{array}{ll}
p_1 = eval(v_1)/F = 0.067099 & p_2 = eval(v_2)/F = 0.019547 \\
p_3 = eval(v_3)/F = 0.050355 & p_4 = eval(v_4)/F = 0.044889 \\
p_5 = eval(v_5)/F = 0.065350 & p_6 = eval(v_6)/F = 0.046677 \\
p_7 = eval(v_7)/F = 0.041315 & p_8 = eval(v_8)/F = 0.046315 \\
p_9 = eval(v_9)/F = 0.041590 & p_{10} = eval(v_{10})/F = 0.054873 \\
p_{11} = eval(v_{11})/F = 0.060372 & p_{12} = eval(v_{12})/F = 0.038712 \\
p_{13} = eval(v_{13})/F = 0.070444 & p_{14} = eval(v_{14})/F = 0.051257 \\
p_{15} = eval(v_{15})/F = 0.077519 & p_{16} = eval(v_{16})/F = 0.061549 \\
p_{17} = eval(v_{17})/F = 0.035320 & p_{18} = eval(v_{18})/F = 0.039750 \\
p_{19} = eval(v_{19})/F = 0.051823 & p_{20} = eval(v_{20})/F = 0.035244
\end{array}
$$

From the values of the probability of a selection we compute the cumulative probability of each chromosome.

Example (cont.)

$$
\begin{array}{llll}
q_1 = 0.067099 & q_2 = 0.086647 & q_3 = 0.137001 & q_4 = 0.181890 \\
q_5 = 0.247240 & q_6 = 0.293917 & q_7 = 0.335232 & q_8 = 0.381546 \\
q_9 = 0.423137 & q_{10} = 0.478009 & q_{11} = 0.538381 & q_{12} = 0.577093 \\
q_{13} = 0.647537 & q_{14} = 0.698794 & q_{15} = 0.776314 & q_{16} = 0.837863 \\
q_{17} = 0.873182 & q_{18} = 0.912932 & q_{19} = 0.964756 & q_{20} = 1.000000
\end{array}
$$

Now we are ready to spin the roulette wheel 20 times.

Each time we select a single chromosome for a new population. Let us assume that a (random) sequence of 20 numbers from the range $[0, 1]$ is:

0.513870	0.175741	0.308652	0.534534	0.947628
0.171736	0.702231	0.226431	0.494773	0.424720
0.703899	0.389647	0.277226	0.368071	0.983437
0.005398	0.765682	0.646473	0.767139	0.780237

The first number $r = 0.513870$ is greater than q_{10} and smaller than q_{11}, meaning the chromosome v_{11} is selected for the new population; the second number $r = 0.175741$ is greater than q_3 and smaller than q_4, meaning the chromosome v_4 is selected for the new population, etc.

The new population consists of the following chromosomes:

Example (cont.)

$$
\begin{aligned}
v'_1 &= (01100111111011010110000110111100 0)\,(v_{11}) \\
v'_2 &= (10001100010110100111100000111001 0)\,(v_4) \\
v'_3 &= (00100010000011010111101101111101 1)\,(v_7) \\
v'_4 &= (01100111111011010110000110111100 0)\,(v_{11}) \\
v'_5 &= (00010101001111111111000011000110 0)\,(v_{19}) \\
v'_6 &= (10001100010110100111100000111001 0)\,(v_4) \\
v'_7 &= (11101110110111000010001111101111 0)\,(v_{15}) \\
v'_8 &= (00011101100101001101011111100010 1)\,(v_5) \\
v'_9 &= (01100111111011010110000110111100 0)\,(v_{11}) \\
v'_{10} &= (00001000001100100000101011101110 1)\,(v_3) \\
v'_{11} &= (11101110110111000010001111101111 0)\,(v_{15}) \\
v'_{12} &= (01000000101110100010110101100111 1)\,(v_9) \\
v'_{13} &= (00010100001001010100101011111101 1)\,(v_6) \\
v'_{14} &= (10000110000111010001011010110011 1)\,(v_8) \\
v'_{15} &= (10111001011001111001100010111111 0)\,(v_{20}) \\
v'_{16} &= (10011010000000111111101001101111 1)\,(v_1) \\
v'_{17} &= (00000111100011000001101000011101 1)\,(v_{10}) \\
v'_{18} &= (11101111101000100011000000100011 0)\,(v_{13}) \\
v'_{19} &= (11101110110111000010001111101111 0)\,(v_{15}) \\
v'_{20} &= (11001111000001111110000110100101 1))v_{16})
\end{aligned}
$$

Example (cont.)

Now we are ready to apply the recombination operator, **crossover**, to the individuals in the new population (vectors v_i').

The probability of crossover $p_c = 0.25$, so we expect that (on average) 25% of chromosomes (i.e., 5 out of 20) undergo crossover.

We proceed in the following way: for each chromosome in the (new) population we generate a random number r from the range $[0, 1]$. If $r < p_c = 0.25$, we select a given chromosome for crossover.

Let us assume that the sequence of random numbers is:

0.822951	0.151932	0.625477	0.314685	0.346901
0.917204	0.519760	0.401154	0.606758	0.785402
0.031523	0.869921	0.166525	0.674520	0.758400
0.581893	0.389248	0.200232	0.355635	0.826927

This means that the chromosomes v_2', v_{11}', v_{13}' and $v_{18'}$ were selected for crossover.

We have been lucky: the number of selected chromosomes is even, so we can pair them easily. If the number of selected chromosomes were odd, we would either add one extra chromosome or remove one selected chromosome - this choice is made randomly as well.

Example (cont.)

Now we **mate selected crhromosomes randomly**: say, the first two (i.e., v_2' and v_{11}') and the next two (i.e., v_{13}' and v_{18}') are coupled together.

For each of these two pairs, we generate a random integer number *pos* from the range [1 : 32] (33 is the total length - number of bits - in a chromosome).

The number *pos* indicates the position of the crossing point. The first pair of chromosomes is

$$\begin{aligned} v_2' &= (100011000101101001111000001110010) \\ v_{11}' &= (111011101011100001000111111011110) \end{aligned}$$

and the generated random number is $pos = 9$. These chromosomes are cut after the 9th bit and replaced by a pair of their offspring:

$$\begin{aligned} v_2'' &= (111011101011010011110000011100 10) \\ v_{11}'' &= (100011000101110000100011111011110) \end{aligned}$$

Example (cont.)

The second pair of chromosomes is

$$\begin{aligned} v'_{13} &= (000101000010010101001010111111011) \\ v'_{18} &= (111011111010001000110000001000110) \end{aligned}$$

and the generated number $pos = 20$. These chromosomes are replaced by a pair of their offspring:

$$\begin{aligned} v''_{13} &= (111011111010001000111010111111011) \\ v''_{18} &= (000101000010010101000000001000110) \end{aligned}$$

The current version of the population is:

Example (cont.)

$$
\begin{aligned}
v_1' &= (01100111111011010110000110111000)\\
v_2'' &= (11101110110110100111100000111001 0)\\
v_3' &= (00100010000011010111011011111011)\\
v_4' &= (01100111111011010110000110111000)\\
v_5' &= (00010101001111111110000110001100)\\
v_6' &= (10001100010110100111100000111001 0)\\
v_7' &= (11101110110111000100011111011110)\\
v_8' &= (00011101100101001101011111000101)\\
v_9' &= (01100111111011010110000110111000)\\
v_{10}' &= (00001000001100100000101011011101)\\
v_{11}'' &= (10001100010111000010001111101111 0)\\
v_{12}' &= (01000000101110100010110101100111)\\
v_{13}' &= (11101111101000100011010111111011)\\
v_{14}' &= (10000110000111010001011010110011 1)\\
v_{15}' &= (10111001011001111001100010111111 0)\\
v_{16}' &= (10011010000000111111101001101111 1)\\
v_{17}' &= (00000111100011000001101000011101 1)\\
v_{18}'' &= (00010100001001010100000000100011 0)\\
v_{19}' &= (11101110110111000100011111011110)\\
v_{20}' &= (11001111000001111110000110100101 1)
\end{aligned}
$$

Example (cont.)

The next operator, **mutation**, is performed on a bit-by-bit basis.

The probability of mutation is $p_m = 0.01$, so we expect that (on average) 1% of bits would undergo mutation.

There are $m \times pop - size = 33 \times 20 = 660$ bits in the whole population, so we expect 6.6 mutations per generation.

Every bit has an equal chance to be mutated, so, for every bit in the population we generate a random number r in the range $[0, 1]$; if $r < 0.01$ we mutate the bit.

This means that we have to generate 660 random numbers. In a sample run, 5 of these numbers were smaller than 0.01. The the random number, the bit number, the chromosome number and the bit number within the chromosome are:

Random num.	Bit num.	Chromosome num.	Bit in chrom.
0.000213	112	4	13
0.009945	349	11	19
0.008809	418	13	22
0.005425	429	13	33
0.002836	602	19	8

Example (cont.)

The current version of the population is:

$$
\begin{aligned}
v_1 &= (01100111111011010110000110111000)\\
v_2 &= (11101110110110100111100000111001 0)\\
v_3 &= (00100010000011010111011011111011)\\
v_4 &= (01100111111011010110000110111000)\\
v_5 &= (00010101001111111110000110001100)\\
v_6 &= (10001100010110100111100000111001 0)\\
v_7 &= (11101110110111000100011111011110)\\
v_8 &= (00011101100101001101011111000101)\\
v_9 &= (01100111111011010110000110111000)\\
v_{10} &= (00001000001100100000101011011101)\\
v_{11} &= (10001100010111000010 0011111011110)\\
v_{12} &= (01000000010111010001011010110011 1)\\
v_{13} &= (00010100001001010100101011111110 1)\\
v_{14} &= (10000110000111010001011010110011 1)\\
v_{15} &= (10111001011001111001100010111111 0)\\
v_{16} &= (10011010000000111111101001101111 1)\\
v_{17} &= (00000111100011000001101000011101 1)\\
v_{18} &= (00010100001001010100000000100011 0)\\
v_{19} &= (11101110110111000100011111011110)\\
v_{20} &= (11001111000001111110000110100101 1)
\end{aligned}
$$

Example (cont.)

We have just completed one iteration (i.e., one generation) of the while loop in the genetic procedure.

Let us examine the results of the evaluation process of the new population.

During the evaluation phase we decode each chromosome and calculate the fitness function values from (x, y) values just decoded. We get:

$$
\begin{array}{ll}
eval(v_1) = 23.410669 & eval(v_2) = 18.201083 \\
eval(v_3) = 16.020812 & eval(v_4) = 23.1412613 \\
eval(v_5) = 20.095903 & eval(v_6) = 17.406725 \\
eval(v_7) = 30.060205 & eval(v_8) = 25.341160 \\
eval(v_9) = 23.410669 & eval(v_{10}) = 19.526329 \\
eval(v_{11}) = 33.351874 & eval(v_{12}) = 16.127799 \\
eval(v_{13}) = 22.692462 & eval(v_{14}) = 17.959701 \\
eval(v_{15}) = 13.666916 & eval(v_{16}) = 26.019600 \\
eval(v_{17}) = 21.278435 & eval(v_{18}) = 27.591064 \\
eval(v_{19}) = 27.608441 & eval(v_{20}) = 23.867227
\end{array}
$$

Note that the total fitness of the new population F is 447.049688, much higher than total fitness of the previous population, 387.776822.

Also, the best chromosome now (v_{11}) has a better evaluation (33.351874) than the best dmmosome (v_{15}) from the previous population (30.060205).

Example (cont.)

After 1000 generations the fitness values of the population are:

$$
\begin{array}{ll}
eval(v_1) = 30.298543 & eval(v_2) = 26.869724 \\
eval(v_3) = 30.316575 & eval(v_4) = 31.933120 \\
eval(v_5) = 30.316575 & eval(v_6) = 34.356125 \\
eval(v_7) = 35.458636 & eval(v_8) = 23.309078 \\
eval(v_9) = 34.393820 & eval(v_{10}) = 30.316575 \\
eval(v_{11}) = 35.477938 & eval(v_{12}) = 35.456066 \\
eval(v_{13}) = 30.316575 & eval(v_{14}) = 32.932098 \\
eval(v_{15}) = 30.746768 & eval(v_{16}) = 34.359545 \\
eval(v_{17}) = 32.932098 & eval(v_{18}) = 32.956664 \\
eval(v_{19}) = 19.669670 & eval(v_{20}) = 32.956664
\end{array}
$$

If we look carefully at the progress during the run, we may discover that in earlier generations the fitness values of some chromosomes were better than the value 35.477938 of the best chromosome after 1000 generations. For example, the best chromosome in generation 396 had value of 38.827553. This is due to the stochastic errors of sampling.

It is relatively easy to keep track of the best individual in the evolution process. It is customary (in genetic algorithm implementations) to store "the best ever"

individual at a separate location; in that way, the algorithm would report the best value found during the whole process (as opposed to the best value in the final population).

Stopping conditions

Some of the various stopping condition are:

- **Maximum generations.** The genetic algorithm stops when the specified number of generation's have evolved.
- **Elapsed time.**The genetic process will end when a specified time has elapsed. Note: If the maximum number of generation has been reached before the specified time has elapsed, the process will end.
- **No change in fitness.**The genetic process will end if there is no change to the population's best fitness for a specified number of generations. *Note:* If the maximum number of generation has been reached before the specified number of generationwith no changes has been reached, the processwill end.
- **Stall generations.**The algorithm stops if there is no improvement in the objective function for a sequence of consecutive generations of length Stall generations.
- **Stall time limit.**The algorithm stops if there is no improvement in the objective function during an interval of time in seconds equal to Stall time limit.

The Fundamental Theorem of Genetic Algorithms

The **Fundamental Theorem of Genetic Algorithm**, also called the Schema theorem that says, in short, that low-order schemata with above-average fitness increase exponentially in successive generations. These particular schema are called building blocks.

Definitions. A **schema** is a template that identifies a subset of strings with similarities at certain string positions.

The **order** of a schema is defined as the number of fixed positions in the template

The **fitness of a schema** is the average fitness of all strings matching the schema.

For example, consider binary strings of length 6.

- The schema

$$1 * 10 * 1$$

  describes the set of all strings of length 6 with 1's at positions 1, 3 and 6 and a 0 at position 4.
- The * is a wildcard symbol, which means that positions 2 and 5 can have a value of either 1 or 0.
- The order of 1*10*1 is 4 and its defining length is $6 - 1 = 5$.

The Fundamental Theorem of Genetic Algorithms

Theorem.
For a given schema H, let:

- $m(H, t)$ be the relative frequency of the schema H in the population of the t^{th} generation.
- $f(H)$ be the mean fitness of the elements of H.
- $O(H)$ be the number of fixed bits in the schema H, called the order of the schema.
- $\delta(H)$ be distance between the first and the last fixed bit of the schema, called the definition length of the schema.
- $\overline{f}$ be the mean fitness of the current population.
- p_c be the crossover probability.
- p_m be the mutation probability.

Then,

$$E[m(H, t+1)] \geq m(H, t)\frac{f(H)}{\overline{f}}\left[1 - p_c\frac{\delta(H)}{} - O(H)p_m\right]$$

Chromosome representations. The Traveling Salesman Problem

The problem: *A traveling salesman must visit every city in his territory exactly once and then return to the starting point.; given the cost of travel between all cit,ies, how should he plan his itinerary for minimum total cost of the entire tour?*

- To solve the problem with a GA, first, we should address an important question connected with the chromosome representation: should we leave a chromosome to be an integer vector, or rather we should transform it into a binary string?
- Until now, we represented a chromosome as a binary vector. This allowed us to use binary mutation and crossover; applying these operators we got legal offspring, i.e., offspring within the search space.
- This is not the case for the traveling salesman problem. In a binary representation of a n cities TSP problem, each city should be coded as a string of $\log_2 n$ bits: if there are 20 cities , we need 5 bits to represent a city.
- A chromosome will be a string of $n \log_2 n$ bits.

The Traveling Salesman Problem (TSP)

- A mutation can result in a sequence of cities which is not a tour, this is: we can get the same city twice in a sequence.
- Moreover, for a TSP with 20 cities some 5-bit sequences (for example, $(10101)_2 = 21$) do not correspond to any city. Similar problems are present when applying crossover operator.
- Clearly, if we use mutation and crossover operators as defined earlier, we would need some sort of a "repair algorithm"; such an algorithm would "repair" a chromosome, moving it back into the search space.
- It seems that the integer vector representation is better: instead of using repair algorithms, we can incorporate the knowledge of the problem into operators.
- In this particular approach we accept integer representation: a vector $v = (i_1, i_2, ..., i_n)$ represents a tour: from i_1 to i_2, etc., from i_{n-1} to i_n and back to i_1, where v is a permutation of $\{12...n\}$.

The Traveling Salesman Problem (TSP)

- For the **initialization** process we can either use some heuristics, or we can initialize the population by a random sample of permutations of $\{12...n\}$.
- The evaluation of a chromosome is straightforward: given the cost of travel between all cities, we can easily calculate the total cost of the entire tour.
- Given two parents, one can construct an offspring by choosing a subsequence of a tour from one parent and preserving the relative order of cities from the other parent. For example, if the parents are

$$(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12) \qquad (3, 7, 1, 11, 4, 12, 5, 2, 10, 9, 6, 8)$$

and the chosen part is

$$(4, 5, 6, 7)$$

the resulting offspring is

$$(1, 11, 4, 5, 6, 7, 12, 2, 10, 9, 8, 3)$$

Chromosome representations. The prisoner's dilemmas

Exercise 9. Use a genetic algorithm to learn a strategy for a game known as the prisoner's dilemma.

- Two members of a criminal gang are arrested and imprisoned.
- Each prisoner is in solitary confinement with no means of communicating with the other.
- They hope to get both sentenced to a year in prison on a lesser charge.
- Simultaneously, the judge offers each prisoner a bargain:
- Each prisoner is given the opportunity either to: betray the other by testifying that the other committed the crime, or to cooperate with the other by remaining silent.
- The offer is:

	Prisoner B silent	Prisoner B betrais A
Prisoner A silent	Both 1 year in prison	B free and A 3 years in prison
Prisoner A betrais B	A free and B 3 years in prison	Each 6 years in prison

The prisoner's dilemma is to decide whether to betray or cooperate with the other prisoner.

Optimization of a function. The prisoner's dilemmas

The prisoner's dilemma can be played as a game between two players, where at each turn, each player either betrais or cooperates with the other prisoner. The players then score according to the payoffs listed in the table.

Player A	Player B	P_1	P_2
Betrais	Betrais	6	6
Betrais	Cooperates	5	0
Cooperates	Betrais	0	5
Cooperates	Cooperates	1	1