Optimization

Màster de Fonaments de Ciència de Dades

Lecture VII. Heuristic optimization methods

Gerard Gómez

Heuristic methods

- For low dimension problems, classical optimization methods usually suffice
- There are many optimization problems, that arise frequently in applications, for which no reasonably fast algorithms have been developed
- For large dimension problems special techniques must be employed. Heuristic methods are among such techniques

Heuristic methods

- **Heuristic** designates a computational procedure that determines an optimal solution by iteratively trying to improve a candidate solution with regard to a given measure of quality
- **Heuristic algorithms** are strategies that "guide" the search process
- Heuristic algorithms are not problem-specific and make few or no assumptions about the problem
- The goal of these methods is to explore large spaces of candidate solutions toward finding optimal or near-optimal solutions at a "reasonable computational cost"
- These kinds of algorithms do not guarantee either feasibility or optimality, or even in many cases to state how close to optimality a particular feasible solution is
- They implement some form of stochastic search optimization
- They may incorporate mechanisms to avoid getting trapped in confined areas of the search space

Heuristic methods

Some heuristic methods:

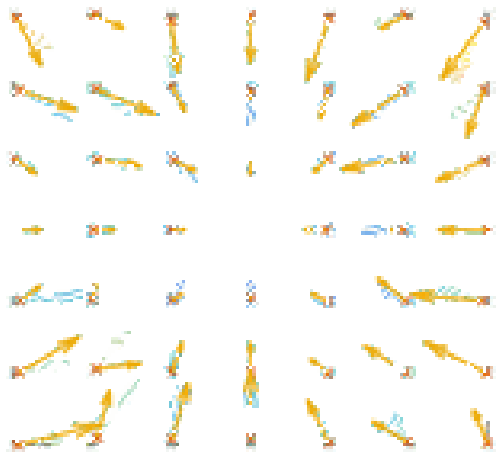
- **Particle Swarm Optimization**
- **Ant colony optimization**
- **Genetic Algorithms**
- Differential evolution
- Adaptive simulated annealing
- Global multi-start method
- **Stochastic methods**
- .....

Pseudocode of a general heuristic algorithm

- **Initialization**
  - Fix the population size M
  - Define the objective (fitness) function f
  - Define the constraints g and h
  - Randomly position the M members of the population in the search space
  - Define a stopping criteria
  - Set the maximum number of iterations $MaxIter$
  - Set $Iter = 1$
- **while** $(Iter \leq MaxIter)$ **do**
  - Execute the algorithm on the population (that, eventually, will increase its size)
  - Evaluate the fitness, $f(\boldsymbol{x})$, of the population
  - Choose the fittest M members and discard the weaker ones
  - If the stopping criteria is satisfied exit, otherwise continue
  - $Iter = Iter + 1$
- **end while**

The member $\boldsymbol{x}$ of the population with the highest fitness value is the (global) optimum solution

Particle Swarm Optimization



Particle Swarm Optimization

- **Particle swarm optimization** (PSO), developed around 1995 by James Kennedy and Russell C. Eberhart, is a heuristic global optimization method
- The **set of candidate solutions** to the optimization problem is defined as
  - a **swarm of particles**
    - which may **flow through the search space defining trajectories**
    - which are **driven by their own and neighbors' best performances**
- The **evolution of the trajectories** is based on **cooperation and competition** among individuals **through generations (iterations)**
- The **flow of information among particles**, which can be limited to a local neighborhood (partial PSO) or extended to the whole swarm (global PSO), is an **essential characteristic** of the algorithm

Particle Swarm Optimization. The basic algorithm

Consider a maximization problem: given $f : D \subset \mathbb{R}^n \to \mathbb{R}$, (fitness function), find $\boldsymbol{x}^*$ such that

$$f(\boldsymbol{x}^*) \geq f(\boldsymbol{x}), \quad \forall \boldsymbol{x} \in D$$

1. **Initialization**

   Select a number M of particles, and for each of the particles

   1.1 Initialize (randomly), for $t = 0$, the position $\boldsymbol{x}_i(0)$, and the velocity $\boldsymbol{v}_i(0)$ for $i = 1, ..., M$

   1.2 Calculate the fitness of each particle $f(\boldsymbol{x}_i(0))$, for $i = 1, ..., M$

   1.3 Initialize the particle's best position to its initial position

   $$\boldsymbol{p}_i(0) = \boldsymbol{x}_i(0), \quad \text{for} \quad i = 1, ..., M$$

   1.4 Initialize the global best position as

   $$\boldsymbol{g}(0) = \boldsymbol{x}_j(0)$$

   if

   $$f(\boldsymbol{x}_j(0)) \geq f(\boldsymbol{x}_k(0)), \quad \text{for all} \quad k = 1, ..., M, \ k \neq j$$

Particle Swarm Optimization. The basic algorithm

2. **Until a stopping criterion is met, repeat the following steps**

   2.1 Update the particle velocity according to

$$\boldsymbol{v}_i(t+1) = w\,\boldsymbol{v}_i(t) + c_1\boldsymbol{R}_1\left[\boldsymbol{p}_i(t) - \boldsymbol{x}_i(t)\right] + c_2\boldsymbol{R}_2\left[\boldsymbol{g}(t) - \boldsymbol{x}_i(t)\right]$$

   where

   - w, c_1 and c_2 are real-valued fixed constants during all the iterations, usually $w \in [0.3, 0.9]$, $0 \leq c_1,\ c_2 \leq 4$
   - $\boldsymbol{R}_1$ and $\boldsymbol{R}_2$ are two diagonal matrices of random numbers, with uniform distribution in [0,1], updated at each step

   2.2 Update the particle position according to

$$\boldsymbol{x}_i(t+1) = \boldsymbol{x}_i(t) + \boldsymbol{v}_i(t+1)$$

   2.3 Evaluate the fitness of each particle

$$f(\boldsymbol{x}_i(t+1)), \quad i = 1, \ldots, M$$

Particle Swarm Optimization. The basic algorithm

2.4 If $f(\boldsymbol{x}_i(t+1)) \geq f(\boldsymbol{p}_i(t))$, update the particle best position:

$$\boldsymbol{p}_i(t+1) = \boldsymbol{x}_i(t+1)$$

2.5 If $f(\boldsymbol{x}_i(t+1)) \geq f(\boldsymbol{g}(t))$, update the global best position:

$$\boldsymbol{g}(t+1) = \boldsymbol{x}_i(t+1)$$

2.6 Test the stopping criterion. If the stopping criterion is satisfied stop, else continue

3. At the end of the iterative process, the best solution is given by

$$\boldsymbol{g}(t+1) \quad \text{and} \quad f(\boldsymbol{g}(t+1))$$

Particle Swarm Optimization. The basic algorithm

Remarks on the velocity update

$$\boldsymbol{v}_i(t+1) = w\,\boldsymbol{v}_i(t) + c_1\boldsymbol{R}_1\left[\boldsymbol{p}_i(t) - \boldsymbol{x}_i(t)\right] + c_2\boldsymbol{R}_2\left[\boldsymbol{g}(t) - \boldsymbol{x}_i(t)\right]$$

- The **inertia weight** w **(usually in** $[0.3, 0.9]$**) reduces the velocities over time** (iterations). If $w \geq 1$ the velocities increase over time and particles can hardly change their direction to move back towards optimum, then the swarm diverges. If $w \ll 1$, then little information is used from the previous step and quick changes of direction appear in the process

- The **constants** c_1 **and** c_2 are called **cognitive coefficient** and **social coefficient**, and modulate the **magnitude of the steps** taken by the particle **in the direction of its personal best** and **the global best**, respectively

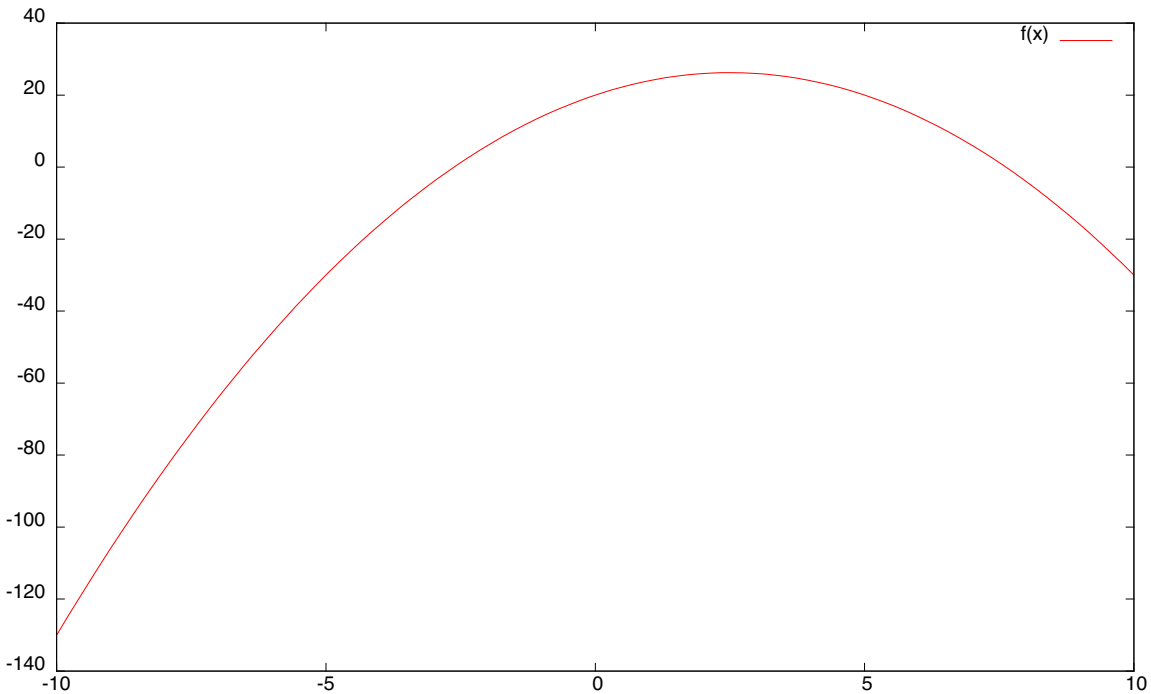
- The two diagonal matrices $\boldsymbol{R}_1$ and $\boldsymbol{R}_2$ of random numbers give a **stochastic influence on both the cognitive and the social components** of the velocity of each particle

  Accordingly, **the trajectories drawn by the particles are semi-random**, as they derive from the contribution of systematic attraction towards the personal and global best solutions, and stochastic weighting of these two acceleration terms

Particle Swarm Optimization

Example. Find the maximum of the function

$$f(x) = -x^2 + 5x + 20 \quad \text{with} \quad -10 \leq x \leq 10$$



- The solution is $x^* = 2.5$, $f(x^*) = 26.25$
- We set $M = 9$, $w = 1$, $c_1 = c_2 = 1$

Example. Initialization $t = 0$

Step 1.1 Initialize the position (randomly) and the velocity of the population at $t = 0$

$$\begin{array}{llllll} x_1(0) = -9.6000, & x_2(0) = -6.0000, & x_3(0) = -2.6000 \\ x_4(0) = -1.1000, & x_5(0) = 0.6000, & x_6(0) = 2.3000 \\ x_7(0) = 2.8000, & x_8(0) = 8.3000, & x_9(0) = 10.0000 \end{array}$$

$$v_1(0) = v_2(0) = v_3(0) = v_4(0) = v_5(0) = v_6(0) = v_7(0) = v_8(0) = v_9(0) = 0$$

Step 1.2 Calculate the fitness of each particle

$$\begin{array}{llllll} f(x_1(0)) = -120.1600, & f(x_2(0)) = -46.0000, & f(x_3(0)) = 0.2400 \\ f(x_4(0)) = 13.2900, & f(x_5(0)) = 22.6400, & f(x_6(0)) = 26.2100 \\ f(x_7(0)) = 26.1600, & f(x_8(0)) = -7.3900, & f(x_9(0)) = -30.0000 \end{array}$$

Step 1.3 Initialize the personal best position of each particle

$$\begin{array}{llllll} p_1(0) = -9.6000, & p_2(0) = -6.0000, & p_3(0) = -2.6000 \\ p_4(0) = -1.1000, & p_5(0) = 0.6000, & p_6(0) = 2.3000 \\ p_7(0) = 2.8000, & p_8(0) = 8.3000, & p_9(0) = 10.0000 \end{array}$$

Step 1.4 Initialize the global best position, which is

$$g(0) = x_6(0) = 2.3000$$

since

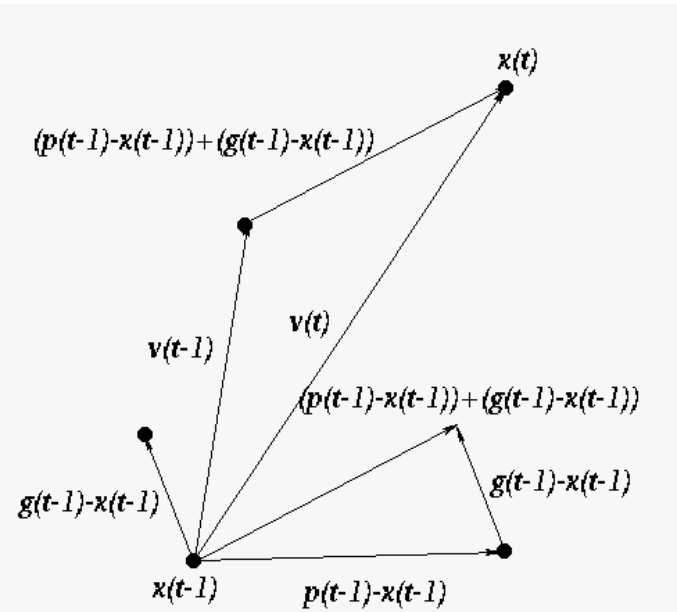
$$f(x_6(0)) \geq f(x_i(0)), \quad \text{for} \quad i = 1, \ldots, 9$$

Example. First iteration $t = 1$

Step 2.1 Compute two random numbers, $R_1 = 0.213$, $R_2 = 0.876$, in $[0, 1]$, and update the particles velocity according to

$$v_i(t) = w\, v_i(t-1) + c_1 R_1 \left[p_i(t-1) - x_i(t-1)\right] + c_2 R_2 \left[g(t-1) - x_i(t-1)\right]$$

$$\begin{array}{rcll}
v_1(1) & = & 0 + 0.213\,(-9.6 + 9.6) + 0.876\,(2.3 + 9.6) & = 10.4244 \\
v_2(1) & = & 0 + 0.213\,(-6 + 6) + 0.876\,(2.3 + 6) & = 7.2708 \\
v_3(1) & = & 0 + 0.213\,(-2.6 + 2.6) + 0.876\,(2.3 + 2.6) & = 4.2924 \\
v_4(1) & = & 0 + 0.213\,(-1.1 + 1.1) + 0.876\,(2.3 + 1.1) & = 2.9784 \\
v_5(1) & = & 0 + 0.213\,(0.6 - 0.6) + 0.876\,(2.3 - 0.6) & = 1.4892 \\
v_6(1) & = & 0 + 0.213\,(2.3 - 2.3) + 0.876\,(2.3 - 2.3) & = 0.0000 \\
v_7(1) & = & 0 + 0.213\,(2.8 - 2.8) + 0.876\,(2.3 - 2.8) & = -0.4380 \\
v_8(1) & = & 0 + 0.213\,(8.3 - 8.3) + 0.876\,(2.3 - 8.3) & = 5.2560 \\
v_9(1) & = & 0 + 0.213\,(10 - 10) + 0.876\,(2.3 - 10) & = -6.7452
\end{array}$$



Example. First iteration $t = 1$

Step 2.2 Update the particles position according to

$$x_i(t) = x_i(t-1) + v_i(t)$$

$$\begin{array}{cccccccccc} x_1(1) & = & 0.8244, & x_2(1) & = & 1.2708, & x_3(1) & = & 1.6924 \\ x_4(1) & = & 1.8784, & x_5(1) & = & 2.0892, & x_6(1) & = & 2.3000 \\ x_7(1) & = & 2.3620, & x_8(1) & = & 3.0440, & x_9(1) & = & 3.2548 \end{array}$$

Step 2.3 Calculate the fitness of each particle $x_i(1)$ using

$$f(x) = -x^2 + 5x + 20$$

$$\begin{array}{cccccccccc} f(x_1) & = & 23.4424, & f(x_2) & = & 24.7391, & f(x_3) & = & 25.5978 \\ f(x_4) & = & 25.8636, & f(x_5) & = & 26.0812, & f(x_6) & = & 26.2100 \\ f(x_7) & = & 26.2310, & f(x_8) & = & 25.9541, & f(x_9) & = & 25.6803 \end{array}$$

Example. First iteration $t = 1$

Step 2.4 Update the personal best for each particle, $i = 1, ..., 9$ according to

$$p_i(1) = \begin{cases} p_i(0) & \text{if} \quad f(x_i(1)) < f(p_i(0)) \\ \\ x_i(1) & \text{if} \quad f(x_i(1)) \geq f(p_i(0)) \end{cases}$$

$$\begin{array}{lllllllll} p_1(1) & = & 0.8244, & p_2(1) & = & 1.2708, & p_3(1) & = & 1.6924 \\ p_4(1) & = & 1.8784, & p_5(1) & = & 2.0892, & p_6(1) & = & 2.3000 \\ p_7(1) & = & 2.3620, & p_8(1) & = & 3.0440, & p_9(1) & = & 3.2548 \end{array}$$

Step 2.5 Find the global best

$$g(1) = x_7(1) = 2.3620$$

Step 2.6 If the stopping criterion is satisfied stop, else continue

Example. Second iteration $t = 2$

Step 2.1 Compute two random numbers $R_1 = 0.113$, $R_2 = 0.706$ in $[0, 1]$, and update the particles velocity according to

$$v_i(t) = w\, v_i(t-1) + c_1 R_1 \left[p_i(t-1) - x_i(t-1)\right] + c_2 R_2 \left[g(t-1) - x_i(t-1)\right]$$

$$\begin{array}{lllllllll} v_1(2) & = & 11.5099, & v_2(2) & = & 8.0412, & v_3(2) & = & 4.7651 \\ v_4(2) & = & 3.3198, & v_5(2) & = & 1.6818, & v_6(2) & = & 0.0438 \\ v_7(2) & = & -0.4380, & v_8(2) & = & -5.7375, & v_9(2) & = & -7.3755 \end{array}$$

Step 2.2 Find the new values of $x_i(2)$ for $i = 1, \ldots, 9$ using

$$x_i(2) = x_i(1) + v_i(2)$$

$$\begin{array}{lllllllll} x_1(2) & = & 12.3343, & x_2(2) & = & 9.3120, & x_3(2) & = & 6.4575 \\ x_4(2) & = & 5.1982, & x_5(2) & = & 3.7710, & x_6(2) & = & 2.3438 \\ x_7(2) & = & 1.9240, & x_8(2) & = & -2.6935, & x_9(2) & = & -4.1207 \end{array}$$

Example. Second iteration $t = 2$

Step 2.3 Calculate the fitness of each particle $x_i(2)$ using

$$f(x) = -x^2 + 5x + 20$$

$$\begin{array}{lllllllll} f_1^2 & = & -70.4644, & f_2^2 & = & -20.1532, & f_3^2 & = & 10.5879 \\ f_4^2 & = & 18.9696, & f_5^2 & = & 24.6346, & f_6^2 & = & 26.2256 \\ f_7^2 & = & 25.9182, & f_8^2 & = & -0.7224, & f_9^2 & = & -17.5839 \end{array}$$

Note that $f(x_7(1)) = 26.2310 > f(x_6(2)) = 26.2256$ (no improving in the global best)

Step 2.4 Update the personal best position for each particle, $i = 1, ..., 9$

$$\begin{array}{lllllllll} p_1(2) & = & 0.8244, & p_2(2) & = & 1.2708, & p_3(2) & = & 1.6924 \\ p_4(2) & = & 1.8784, & p_5(2) & = & 2.0892, & p_6(2) & = & 2.3438 \\ p_7(2) & = & 2.3620, & p_8(2) & = & 3.0440, & p_9(2) & = & 3.2548 \end{array}$$

Step 2.5 Find the global best position

$$g(2) = x_7(1) = 2.3620$$

Step 2.6 If the stopping criterion is satisfied stop, else continue

Example. Third iteration $t = 3$

Step 2.1 Compute two random numbers $R_1 = 0.178$, $R_2 = 0.507$ in $[0, 1]$, and update the particles velocity according to

$$v_i(t) = w\, v_i(t-1) + c_1 R_1 \left[p_i(t-1) - x_i(t-1)\right] + c_2 R_2 \left[g(t-1) - x_i(t-1)\right]$$

$$\begin{array}{lllllllll} v_1(3) & = & 4.4052, & v_2(3) & = & 3.0862, & v_3(3) & = & 1.8405 \\ v_4(3) & = & 1.2909, & v_5(3) & = & 0.6681, & v_6(3) & = & 0.0530 \\ v_7(3) & = & -0.1380, & v_8(3) & = & -2.1531, & v_9(3) & = & -2.7759 \end{array}$$

Step 2.2 Find the new values of $x_i(3)$ for $i = 1, \ldots, 9$ using

$$x_i(3) = x_i(2) + v_i(3)$$

$$\begin{array}{lllllllll} x_1(3) & = & 16.7395, & x_2(3) & = & 12.3982, & x_3(3) & = & 8.2980 \\ x_4(3) & = & 6.4892, & x_5(3) & = & 4.4391, & x_6(3) & = & 2.3968 \\ x_7(3) & = & 1.7860, & x_8(3) & = & -4.8466, & x_9(3) & = & -6.8967 \end{array}$$

Example. Third iteration $t = 3$

Step 2.3 Calculate the fitness of each particle $x_i(2)$ using

$$f(x) = -x^2 + 5x + 20$$

$$\begin{array}{lllllllll} f_1(3) & = & -176.5145, & f_2(3) & = & -71.7244, & f_3()3 & = & -7.3673 \\ f_4(3) & = & 10.3367, & f_5(3) & = & 22.4900, & f_6(3) & = & 26.2393 \\ f_7(3) & = & 25.7402, & f_8(3) & = & -27.7222, & f_9(3) & = & -62.0471 \end{array}$$

Step 2.4 Update the personal best position for each particle, $i = 1, ..., 9$

$$\begin{array}{lllllllll} p_1(3) & = & 0.8244, & p_2(3) & = & 1.2708, & p_3(3) & = & 1.6924 \\ p_4(3) & = & 1.8784, & p_5(3) & = & 2.0892, & p_6(3) & = & 2.3968 \\ p_7(3) & = & 2.3620, & p_8(3) & = & 3.0440, & p_9(3) & = & 3.2548 \end{array}$$

Step 2.5 Find the global best position

$$g(3) = x_6(3) = 2.3968$$

Step 2.6 If the stopping criterion is satisfied stop, else continue

Example. Iterations $t = 4, \ldots, 8$

```
Iteration            4
x   -2.8614   -1.2849    0.2040    0.8609    1.6054    2.4498**  2.5688    4.9774    5.7218
v  -19.6009  -13.6831   -8.0940   -5.6282   -2.8337    0.0530    0.7828    9.8240   12.6185
f   -2.4945   11.9247   20.9786   23.5634   25.4497   26.2475** 26.2453   20.1126   15.8698
p    0.8244    1.2708    1.6924    1.8784    2.0892    2.4498    2.5688    3.0440    3.2548
g    2.4498**
Iteration            5
x  -14.9298   -9.7061   -4.7727   -2.5961   -0.1294    2.5028**  3.2626   11.0434   13.5101
v  -12.0684   -8.4212   -4.9767   -3.4571   -1.7348    0.0530    0.6938    6.0660    7.7883
f -277.5471 -122.7393  -26.6417    0.2793   19.3362   26.2500** 25.6685  -46.7398  -94.9730
p    0.8244    1.2708    1.6924    1.8784    2.0892    2.5028    2.5688    3.0440    3.2548
g    2.5028**
Iteration            6
x  -12.8315   -8.2265   -3.8773   -1.9586    0.2160    2.5558    3.3363   10.0656   12.2402
v    2.0983    1.4796    0.8954    0.6376    0.3454    0.0530    0.0737   -0.9778   -1.2699
f -208.8041  -88.8073  -14.4201    6.3712   21.0334   26.2469   25.5506  -30.9884  -68.6212
p    0.8244    1.2708    1.6924    1.8784    2.0892    2.5028    2.5688    3.0440    3.2548
g    2.5028** (no improvement)
Iteration            7
x   -9.6439   -5.9889   -2.5370   -1.0141    0.7119    2.6046    3.3489    8.5294   10.2554
v    3.1876    2.2376    1.3403    0.9445    0.4958    0.0488    0.0126   -1.5362   -1.9848
f -121.2237  -45.8116    0.8787   13.9011   23.0526   26.2390   25.5293  -10.1040  -33.8960
p    0.8244    1.2708    1.6924    1.8784    2.0892    2.5028    2.5688    3.0440    3.2548
g    2.5028** (no improvement)
Iteration            8
x    1.3342    1.6746    1.9961    2.1379    2.2987    2.5833    2.8012    3.0268    3.1875
v   10.9781    7.6635    4.5331    3.1520    1.5868   -0.0214   -0.5477   -5.5026   -7.0678
f   24.8909   25.5687   25.9961   26.1189   26.2095   26.2431   26.1593   25.9725   25.7773
p    1.3342    1.6746    1.9961    2.1379    2.2987    2.5028    2.5688    3.0268    3.1875
g    2.5028** (no improvement)
```

Example. Iterations $t = 9, \ldots, 13$

```
Iteration            9
x   12.4679    9.4484    6.5967    5.3386    3.9127    2.5336    2.1631   -2.5456   -3.9715
v   11.1337    7.7738    4.6006    3.2006    1.6140   -0.0496   -0.6381   -5.5724   -7.1590
f  -73.1086  -22.0303    9.4672   18.1926   24.2543   26.2489   26.1365    0.7917  -15.6301
p    1.3342    1.6746    1.9961    2.1379    2.2987    2.5028    2.5688    3.0268    3.1875
g    2.5028** (no improvement)
Iteration           10
x    9.7213    7.5354    5.4710    4.5602    3.5280    2.4443    2.0217   -1.1474   -2.1796
v   -2.7466   -1.9130   -1.1257   -0.7784   -0.3847   -0.0893   -0.1414    1.3982    1.7919
f  -25.8966    0.8949   17.4234   22.0057   25.1933   26.2469   26.0213   12.9467    4.3514
p    1.3342    1.6746    1.9961    2.1379    2.2987    2.5028    2.5688    3.0268    3.1875
g    2.5028** (no improvement)
Iteration           11
x   -1.5398   -0.3197    0.8325    1.3409    1.9170    2.4198    2.4425    4.5266    5.1027
v  -11.2610   -7.8551   -4.6384   -3.2193   -1.6109   -0.0245    0.4208    5.6739    7.2823
f    9.9303   18.2991   23.4696   24.9065   25.9101   26.2436   26.2467   22.1430   19.4760
p    1.3342    1.6746    1.9961    2.1379    2.2987    2.5028    2.4425    3.0268    3.1875
g    2.5028** (no improvement)
Iteration           12
x  -10.1234   -6.3057   -2.7000   -1.1093    0.6935    2.4506    2.9025    8.8592   10.6620
v   -8.5836   -5.9859   -3.5326   -2.4502   -1.2235    0.0308    0.4600    4.3326    5.5593
f -133.0994  -51.2895   -0.7903   13.2229   22.9866   26.2476   26.0880  -14.1891  -40.3680
p    1.3342    1.6746    1.9961    2.1379    2.2987    2.5028    2.4425    3.0268    3.1875
g    2.5028** (no improvement)
Iteration           13
x   -7.2226   -4.2880   -1.5165   -0.2938    1.0920    2.5320    2.9361    7.3687    8.7544
v    2.9008    2.0176    1.1835    0.8155    0.3985    0.0814    0.0336   -1.4905   -1.9076
f  -68.2790  -19.8276   10.1175   18.4447   24.2675   26.2490   26.0598    2.5462  -12.8678
p    1.3342    1.6746    1.9961    2.1379    2.2987    2.5028    2.4425    3.0268    3.1875
g    2.5028** (no improvement)
```

Example. Last iterations $t = 14, \ldots, 18$

New values for the velocity computation: w=0.5, $c_1 = 0.5$, $c_2 = 0.9$

```
Iteration           14
 x    3.0562    2.7475    2.6576    2.6180    2.5875    2.4924**  2.4811    2.2615    2.1860
 v    0.1815   -0.0230   -0.0145   -0.0107   -0.0307    0.0174    0.0374   -0.0953   -0.1216
 f   25.9406   26.1888   26.2252   26.2361   26.2424   26.2499**   26.2496   26.1931   26.1514
 p    2.8747    2.7475    2.3535    2.4079    2.5142    2.4925    2.4973    2.3568    2.3076
 g    2.4924**
 Iteration           15
 x    2.6310    2.5420    2.4512    2.4744    2.4986**  2.5316    2.5447    2.4739    2.4590
 v   -0.4252   -0.2055   -0.2064   -0.1436   -0.0888    0.0392    0.0636    0.2124    0.2730
 f   26.2328   26.2482   26.2476   26.2493   26.2500** 26.2490   26.2480   26.2493   26.2483
 p    2.6310    2.5420    2.4512    2.4744    2.4986    2.4925    2.4973    2.4739    2.4590
 g    2.4986**
 Iteration           16
 x    2.3569    2.4304    2.3929    2.4338    2.4711    2.5308    2.5447    2.6116    2.6358
 v   -0.2742   -0.1116   -0.0583   -0.0406   -0.0276   -0.0008   -0.0000    0.1377    0.1768
 f   26.2295   26.2452   26.2385   26.2456   26.2492   26.2490   26.2480   26.2375   26.2316
 p    2.6310    2.5420    2.4512    2.4744    2.4986    2.4925    2.4973    2.4739    2.4590
 g    2.4986** (no improvement)
 Iteration           17
 x    2.3813    2.4503    2.4336    2.4620    2.4876    2.5151    2.5210    2.5996    2.6204
 v    0.0245    0.0199    0.0407    0.0283    0.0165   -0.0157   -0.0237   -0.0120   -0.0154
 f   26.2359   26.2475   26.2456   26.2486   26.2498   26.2498   26.2496   26.2401   26.2355
 p    2.3813    2.5420    2.4512    2.4744    2.4986    2.4925    2.4973    2.4739    2.4590
 g    2.4986** (no improvement)
 Iteration           18
 x    2.4130    2.5130    2.4745    2.4906    2.5062*   2.4983*   2.4990*   2.5256    2.5253
 v    0.0316    0.0627    0.0410    0.0285    0.0187   -0.0168   -0.0220   -0.0740   -0.0951
 f   26.2424   26.2498   26.2494   26.2499   26.2500*  26.2500*  26.2500*  26.2493   26.2494
 p    2.4130    2.5130    2.4745    2.4906    2.4986    2.4983    2.4990    2.5256    2.5253
 g    2.4990**
```

Particle Swarm Optimization. Additional comments

1. **The stopping criterion**

   The stopping criterion mainly depends on the problem and can be:

   1.1 A prespecified total number of iterations

   1.2 A maximum number of iterations since the last update of global best

   1.3 A maximum CPU time

   1.4 A predefined target value of the fitness function

2. **Position and velocity initialization**

   2.1 PSO requires an initial estimate of the positions $\boldsymbol{x}_i = (x_{i,1}, ..., x_{i,n})$ and velocities $\boldsymbol{v}_i = (v_{i,1}, ..., v_{i,n})$ of the M particles $(i = 1, ..., M)$

   2.2 The way $\boldsymbol{x}_i$ and $\boldsymbol{v}_i$ are initialized has an important role in the probability that particles travel outside the feasible set

   2.3 The best option is that the initial positions of the particles cover as uniformly as possible the feasible set

   $$x_{i,j}(0) \sim U(x_{j,min}, x_{j,max}), \quad i = 1, ..., M \quad j = 1, ..., n$$

   2.4 A good option to set the initial velocities to zero, or to very small random numbers, as the exploration ability is still guaranteed by the choice of the initial positions

Particle Swarm Optimization. Additional comments

3. **Choice of the inertia weight** w

   3.1 The inertia weight can be implemented either as a fixed value or dynamically changing values (which is much better)

   3.2 Usually, the inertia value is high at first, which allows all particles to move freely in the search space at the initial steps, and decreases over time

   3.3 Usually, the inertia weight value decreases linearly with the iteration number t according to

$$w^{t+1} = w_{max} - \frac{w_{\max} - w_{\min}}{t_{\max}} t$$

   with $w_{\max} \approx 0.9$ and $w_{\min} \approx 0.3$

4. **Choice of the acceleration constants** c_1 **and** c_2

   4.1 c_1 and c_2 govern the extent to which the particles move towards the individual and global best particle, modulating the relative contributions of the social and cognitive terms

   4.2 In general, it has been shown that the conditions

$$c_1 = c_2 = 2$$

   work well for most of the applications

Particle Swarm Optimization. Additional comments

5. Avoiding the velocity explosion

5.1 If we take $c_1 = c_2 = 2$, then both the terms $c_1 \boldsymbol{R}_1$ and $c_2 \boldsymbol{R}_2$ will be uniformly distributed in [0, 2] with average value equal to 1. As a consequence, it may happen that the trajectory of a particle crosses the boundaries of the feasible set

5.2 To avoid this situation, a velocity threshold of the velocity components is introduced in the algorithm

$$v_j^{\max} = k \frac{x_j^{\max} - x_j^{\min}}{2}, \quad j = 1, ..., n, \quad k \in (0, 1]$$

so that, for $i = 1, ..., N$ and $j = 1, ..., n$

$$\begin{array}{llll} \text{if} & v_{i,j} > v_j^{\max} & \text{then} & v_{i,j} = v_j^{\max} \\ \text{if} & v_{i,j} < -v_j^{\max} & \text{then} & v_{i,j} = -v_j^{\max} \end{array}$$

Particle Swarm Optimization. Additional comments

5. **Swarm population**

   5.1 The size M of the population is another factor that has an impact on the performances of the PSO algorithm

   5.2 A large population increases the computational efforts, but also the diversity of the swarm and its exploration ability. It also increases the probability of premature convergence

   5.3 In most cases it has been demonstrated that when the number of individuals is larger than 50, PSO is not sensitive to the size of the population

Particle Swarm Optimization. Additional comments

6. **Network topology**

   6.1 The basic PSO algorithm may be easily trapped in a local optimum

   6.2 Indeed, fast convergence is often achieved as all the particles tend to be attracted simultaneously to the portion of the search space where the global best is

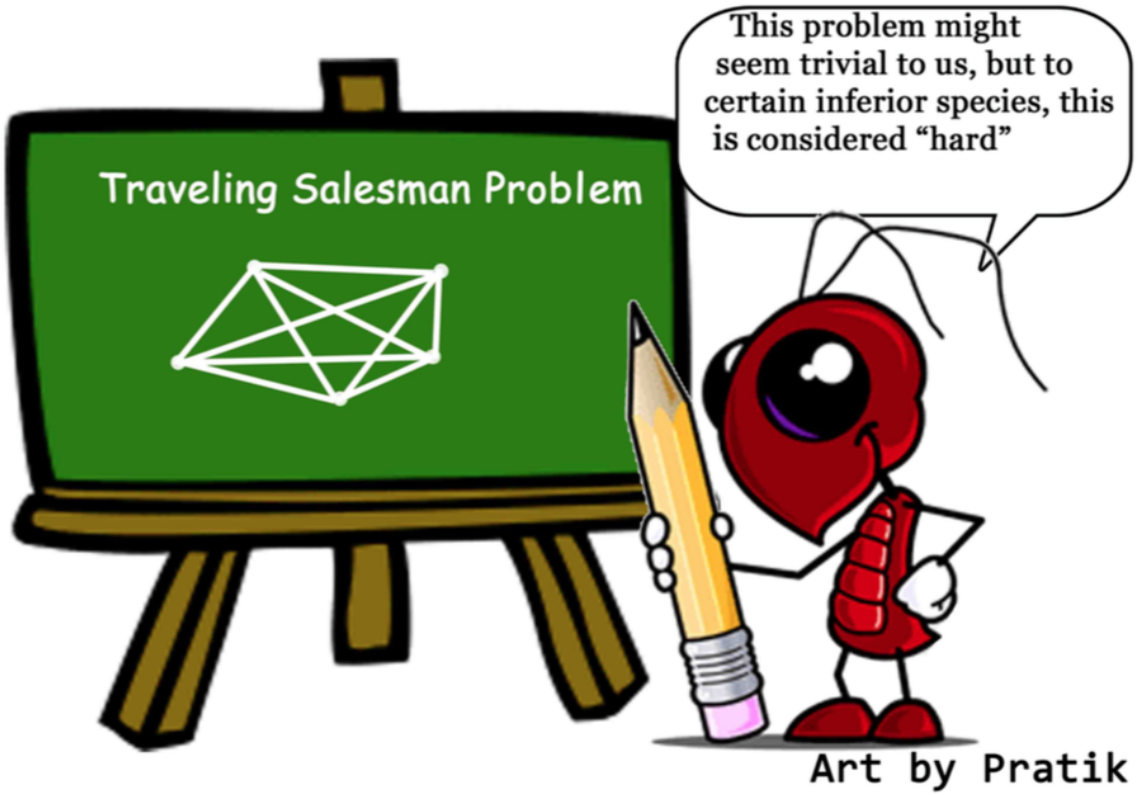
   6.3 If the global optimum is not close to the best particle, this characteristic may hinder the possibility of the swarm to explore other areas

   6.4 One way of limiting the probability of a premature convergence to local optima is to define the social component of the velocity update equation not in terms of the global best $\boldsymbol{g}$ but just based on the best known position $\boldsymbol{l}$ (local best) of a sub-swarm "around" the particle that is moved

   6.5 The advantage of a local best swarm (partial PSO) is that while neighbors are closely connected, the individuals that are topologically distant are also relatively independent of one another, so they may search different portions of the feasible set or explore different local optima without the overall swarm being trapped in any of them

Ant colony optimization

Ant colony optimization

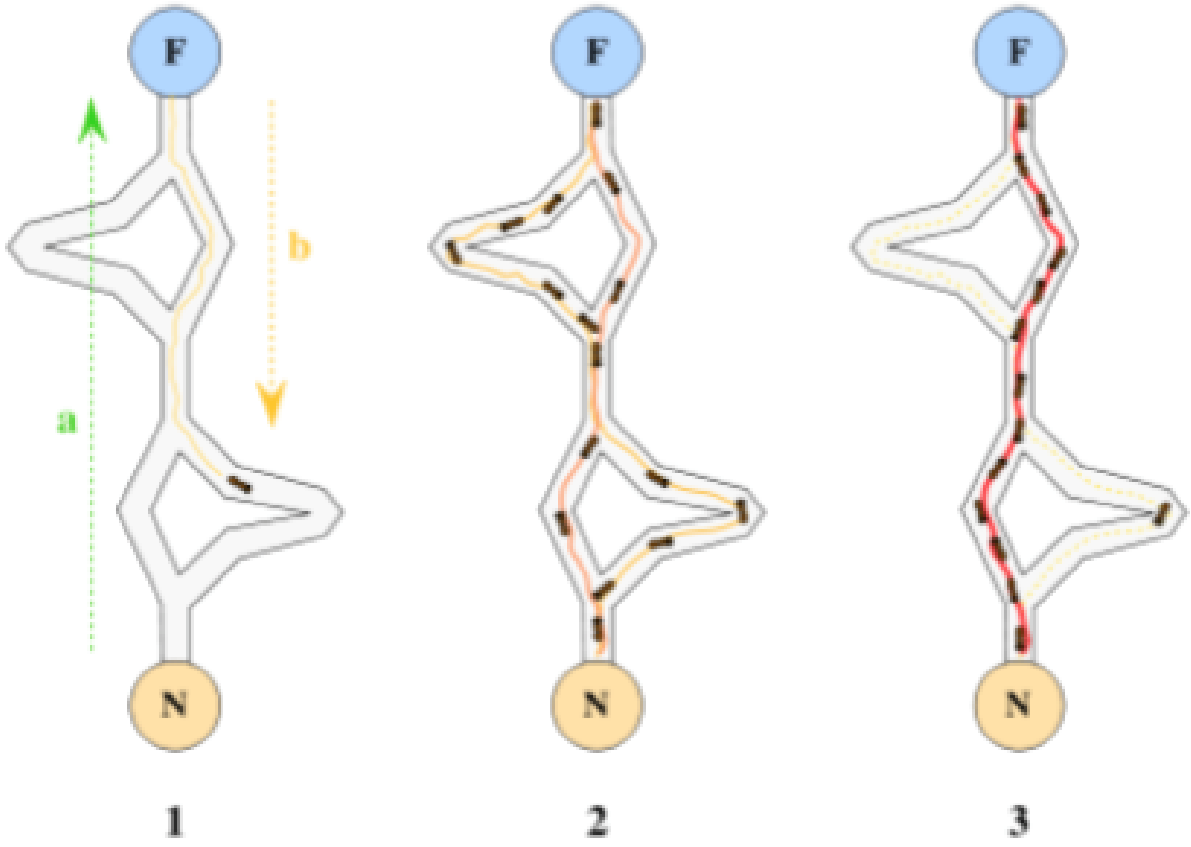


Ant colony optimization algorithm is a technique for solving problems which can be reduced to **finding good paths through graphs**. This method was introduced by Marco Dorigo in 1992

Ant colony optimization

- Shortest path is discovered by ants via **pheromone trails**
- Ants are blind and navigate from nest to food source
- While moving, **ants leave a chemical pheromone trail on the ground**
- Ants **can smell pheromone**
- When choosing their way, they **tend to choose, in probability, paths marked by strong pheromone concentrations**
- It is assumed that, as soon as an ant finds a food source, it evaluates the quantity and the quality of the food and carries some of it back to the nest, and also how long was the path
- During the **return trip**, the **quantity of pheromone that an ant leaves on the ground may depend on the quantity and quality of the food, as well as the length of the path**
- The **pheromone trails will guide other ants** to the food source

Ant colony optimization: the length of the path



A colony of ants has several paths to go from the nest **N** to the food **F**. After some time, almost all endup using the shortest one

Ant colony optimization. Path selection

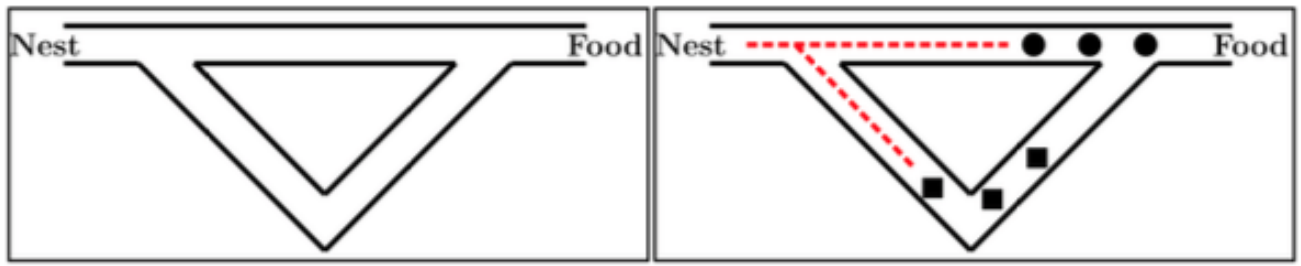
- Pheromones accumulate on path segments
- A path segment is selected at random based on the amount of "trail" present on possible paths from each node
- When an ant reaches next node, selects next path segment
- Continues until reaches the final node
- The finished tour is a candidate solution
- Tour is analyzed for optimality

Ant colony optimization. A simplified model

Consider the following simple model defined by a graph

$$G = (D, P)$$

- D consists of two nodes, namely N (representing the nest of the ants), and F (representing the food source)
- P consists of two paths, namely p_1 and p_2, between N and F
- To p_1 we assign a length of l_1, and to p_2 a length l_2, such that $l_2 > l_1$



Initially, 50% of the ants take the short path p_1 and 50% the long path p_2

- Since ants deposit pheromone on the paths on which they move, the chemical pheromone trails must be modeled
- We introduce an artificial pheromone value τ_i for each of the two paths p_1 and p_2 indicating the strength of the pheromone trail on the corresponding path

Ant colony optimization. A simplified model

- We introduce a certain number M of artificial ants
- Each ant behaves as follows: Starting from N (i.e., the nest), an ant chooses with probability

$$prob_1 = \frac{\tau_1}{\tau_1 + \tau_2}, \quad prob_2 = \frac{\tau_2}{\tau_1 + \tau_2}$$

between path p_1 and path p_2 for reaching the food source F

- Obviously, if $\tau_1 > \tau_2$, the probability of choosing p_1 is higher, and vice versa
- For returning from F to N, an ant **uses the same path it choosed to reach** F, and it changes the artificial pheromone value associated to the used path according to

$$\tau_1 \to \tau_1 + \frac{Q}{l_1}, \quad \tau_2 \to \tau_2 + \frac{Q}{l_2}$$

where the positive constant Q is a parameter of the model. In other words, **the amount of artificial pheromone that is added depends on the length of the chosen path**: the shorter the path, the higher the amount of added pheromone

Ant colony optimization. A simplified model

This model is iteratively simulated as follows:

- At each step (iteration) all the ants are initially placed in node N
- Each ant moves from N to F as outlined above

The circles arrive earlier to F, therefore, when returning, the probability to take again the short path is higher The pheromone trail on the short path receives, in probability, a stronger reinforcement, and the probability to take this path grows. Due to the evaporation of the pheromones, the whole colony will, in probability, use the short path

- The pheromone evaporation in the model is simulated as follows:

$$\tau_1 \to (1-\rho)\tau_1, \quad \tau_2 \to (1-\rho)\tau_2$$

where $\rho \in (0,1)$ is a parameter that regulates the pheromone evaporation

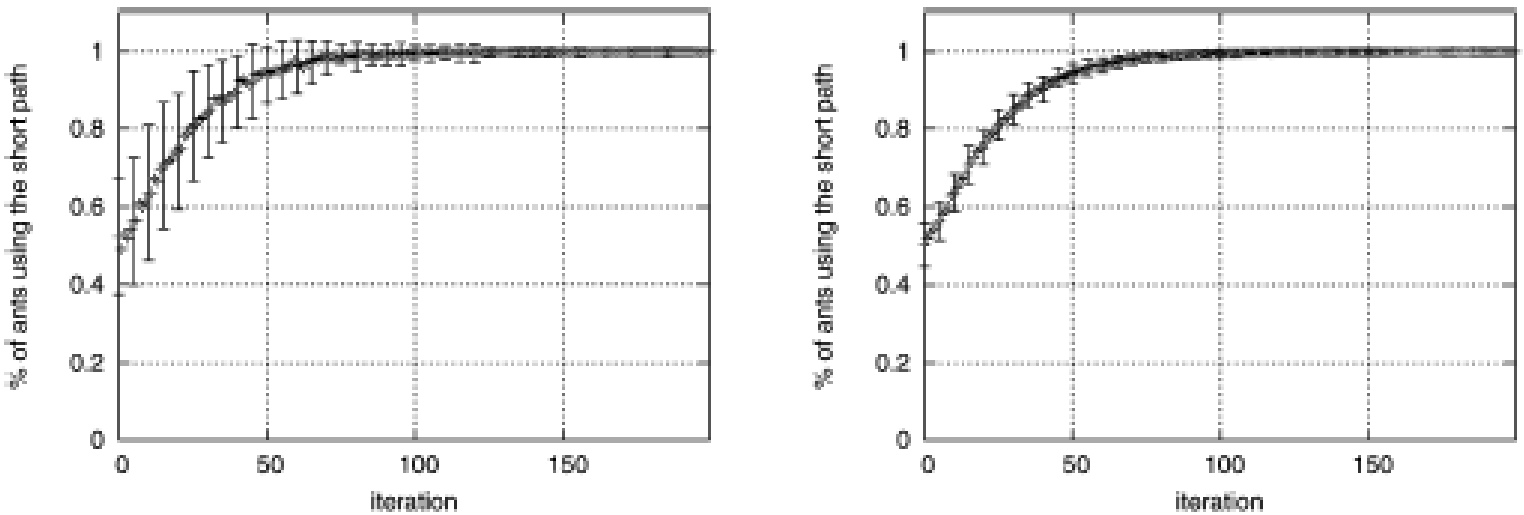
Differences between real ants and the ones of the model

- While **real ants move in their environment in an asynchronous way,** at each iteration of the **simulated system each of the artificial ants moves from the nest to the food source and follows the same path back**
- While real ants leave pheromone on the ground whenever they move, **artificial ants only deposit artificial pheromone on their way back to the nest**
- The foraging behavior of real ants is based on an *implicit evaluation of a solution* (i.e., a path from the nest to the food source)

  By *implicit solution evaluation* we mean the fact that **shorter paths will be completed earlier than longer ones**, and therefore they will receive pheromone reinforcement more quickly

Ant colony optimization. Numerical results

Using $l_1 = 1$, $l_2 = 2$, $Q = 1$, $\tau_1 = \tau_2 = 0.5$ and $\rho = 0$, the figure shows the results of the simulation using $M = 10$ (left) and $M = 100$ (right) initial ants



The x-axis shows the iterations, and the y-axis the percentage of the ants using the short path, the error bars show the standard deviation for each 5th iteration[1]

[1] C. Blum, Physics of Life Reviews 2 (2005) 353–373

Ant colony optimization pseudocode

- **Initialization**
  - Define the components of the graph (nodes and edges), and number of ants M
  - Define the fitness objective function f
  - Fix the pheromone update model (transition probability $prob_i$, and deposit evaporation ρ)
  - Set the termination condition
- **While (termination criteria not met) do**
  - **Construct ant solutions**
    - Every ant builds up a solution vector $\boldsymbol{x}$ using the components of the graph
    - Component solutions are chosen based on the transition probabilities $prob_i$
  - **Pheromone update**
    - Pheromone update: evaporation and increase/decrease of concentrations done based on the solution vectors $\boldsymbol{x}$ found by each ant
  - **Optional actions**
- **End while**

Ant colony optimization algorithm

- To implement the procedure, we need a graph $G = (D, P)$, where D is the set of nodes and P the set of links between them
- Each arc (i, j) of the graph has an associated variable τ_{ij} called the pheromone trail
- The intensity of the pheromone is an indicator of the utility of that arc to build better solutions
- At each node, stochastic decisions are taken to decide on the next node
- Initially, a constant amount of pheromone (i.e., $\tau_{ij} = 1$, for all $i, j \in N$) is allocated to all the arcs
- The probability of the kth ant at node i choosing node j using the pheromone trail τ_{ij} is given by

$$prob_{ij}^{(k)} = \begin{cases} \dfrac{\tau_{ij}^{\alpha}}{\sum_{h \in N_i^{(k)}} \tau_{ih}^{\alpha}} & \text{if} \quad j \in N_i^{(k)} \\ 0 & \text{if} \quad j \notin N_i^{(k)} \end{cases}$$

where $N_i^{(k)}$ is the neighbourhood of ant k when sitting at the ith node, and α is a parameter that controls the influence of τ_{ij}

Ant colony optimization algorithm

- The neighbourhood N_i^k of the ith node contains all nodes directly connected to it excepting the predecessor node. This ensures unidirectional movement of the ants. As an exception for the destination node, where N_i^k should be null, the predecessor of node i is included
- The pheromone level at each iteration is updated by

$$\tau_{ij}(k+1) = \rho\tau_{ij}(k) + \Delta\tau_{ij}(k)$$

where $0 \leq \rho < 1$ and $1 - \rho$ represent the pheromone evaporation rate, and $\Delta\tau_{ij}$ is related to the performance of each ant

Minimizing functions with ant colony optimization

Assume we want to minimize

$$f : [a, b] \subset \mathbb{R} \to \mathbb{R}$$

Inizialitation

- Divide equally $[a, b]$ into M disjoint subintervals of length $\delta = \dfrac{b-a}{M}$ so

$$[a, b] = I_1 \cup I_2 \cup \cdots \cup I_M \quad \text{with} \quad I_i = [a + (i-1)\delta, a + i\delta]$$

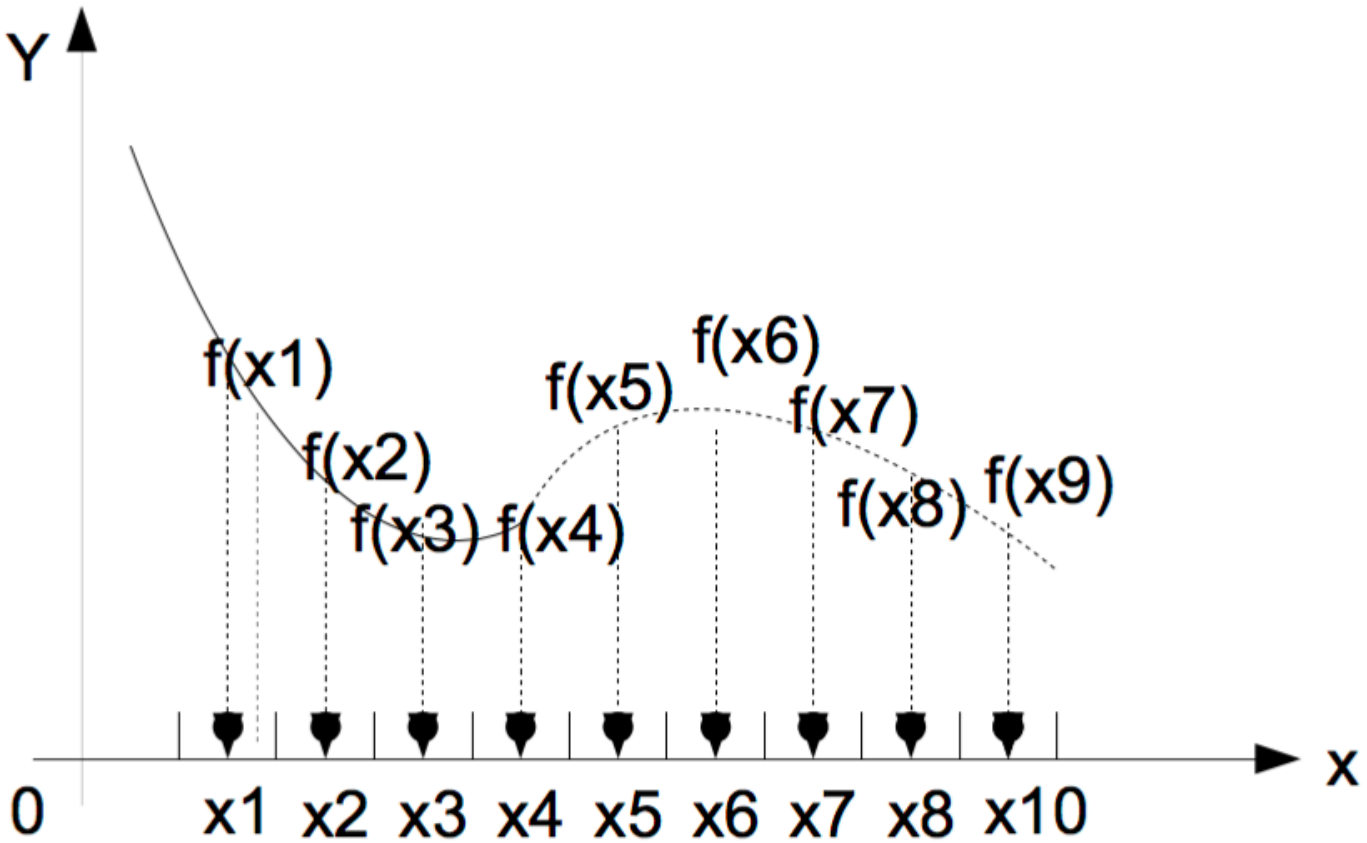
- Denote by x_i the midpoint of I_i

$$x_i = a + \left(i - \frac{1}{2}\right)\delta$$

- At each step of the procedure we start putting an ant at each x_i, and assign to each ant the function value $f(x_i)$; so we have as many ants as intervals I_i

Minimizing functions with ant colony optimization

Inizialitation



Minimizing functions with ant colony optimization

Inizialitation

- Denote by $\tau_i(t)$ the pheromone assigned subinterval I_i at time t. At $t = 0$ all the subintervals have the same quantity of pheromone

$$\tau_i(0) = const > 0$$

- Initialize the increment of pheromones $\Delta\tau_i(0) = 0,\ i = 1, 2, ..., M$

Rule of move of the ants

- Let $\mathcal{N}(I_i)$ be the set of neighborhoods of I_i, then

$$\mathcal{N}(I_i) = \begin{cases} I_2 & \text{if} \quad i = 1 \\ I_{i-1} \cup I_{i+1} & \text{if} \quad i = 2, ..., M-1 \\ I_{M-1} & \text{if} \quad i = M \end{cases}$$

- The ant at I_i will move to its neighborhood according to the weight of virtual edge $E(I_i, I_{i+1})$, which is defined as $f(x_i) - f(x_{i+1})$

- **Assume that the ant a_k is currently at the subinterval I_i, if $f(x_i) - f(x_j) > 0$, the ant a_k will move to its neighbor I_j, otherwise, it will not move**

Rule of move of the ants(cont.)

- Let $\mathcal{N}_i^{(k)}$ be the set of subintervals where the ant a_k can move at a certain step when it is at I_i
- The ant a_k moves from I_i to I_j, according to the probability $p_{ij}^{(k)}(t)$
- We denote by $p_{ij}^{(k)}(t)$ as the probability with which the ant a move from I_i to I_j at the t-th iteration, which is given by

$$p_{ij}^{(k)}(t) = \begin{cases} \dfrac{\tau_{ij}^{\alpha}(t)}{\sum_{h \in \mathcal{N}_i^{(k)}} \tau_{ih}^{\alpha}(t)} & \text{if} \quad j \in \mathcal{N}_i^{(k)} \\ 0 & \text{if} \quad j \notin \mathcal{N}_i^{(k)} \end{cases}$$

  where α is a parameter that controls the influence of τ_{ij}

- **The ant a_k moves from I_i to I_j, according to the probability $p_{ij}^{(k)}(t)$**

Minimizing functions with ant colony optimization

Pheromone update

- If the ant a_k moves from I_i to I_j, according to the probability $p_{ij}^{(k)}(t)$, it will leave pheromone in the subinterval I_j
- **The quantity of pheromone it leaves is** $\Delta\tau_j^k(t) = C\left[f(x_i) - f(x_j)\right]$ (where C is a positive constant) so the greater the value of $f(x_i) - f(x_j)$ is the more pheromone the ant a_k will leave, and the more probable the subinterval I_j will appeal to other ants
- **All the ants that move to I_j will leave there pheromone**; if there are q ants moving to I_j at the t-th iteration, the total pheromone they leave is

$$\Delta\tau_j(t) = \sum_{p=1}^{q} \Delta_j^{j_p}(t)$$

- When all the ants finish the iteration, the **updated value of** $\tau_j(t+1)$ is

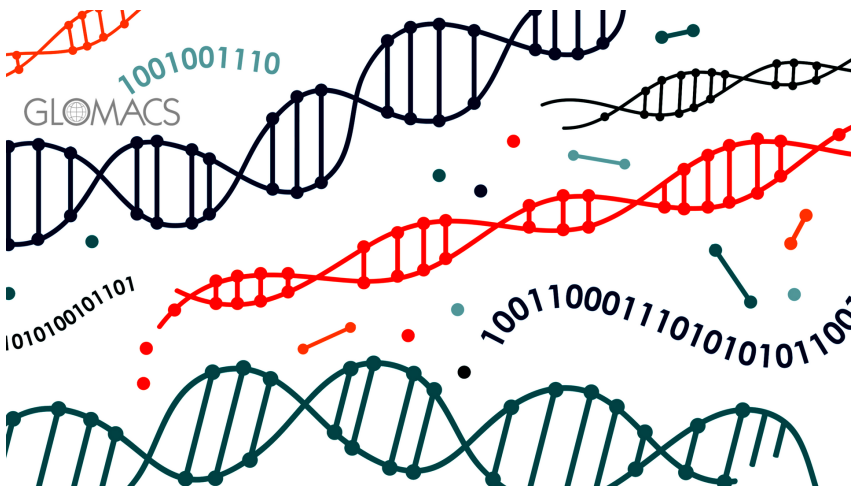
$$\tau_j(t+1 = (1-\rho)\tau_j(t) + \Delta\tau_j(t)$$

where ρ is the coefficient of evaporation and $1-\rho$ is the coefficient for the remaining pheromone

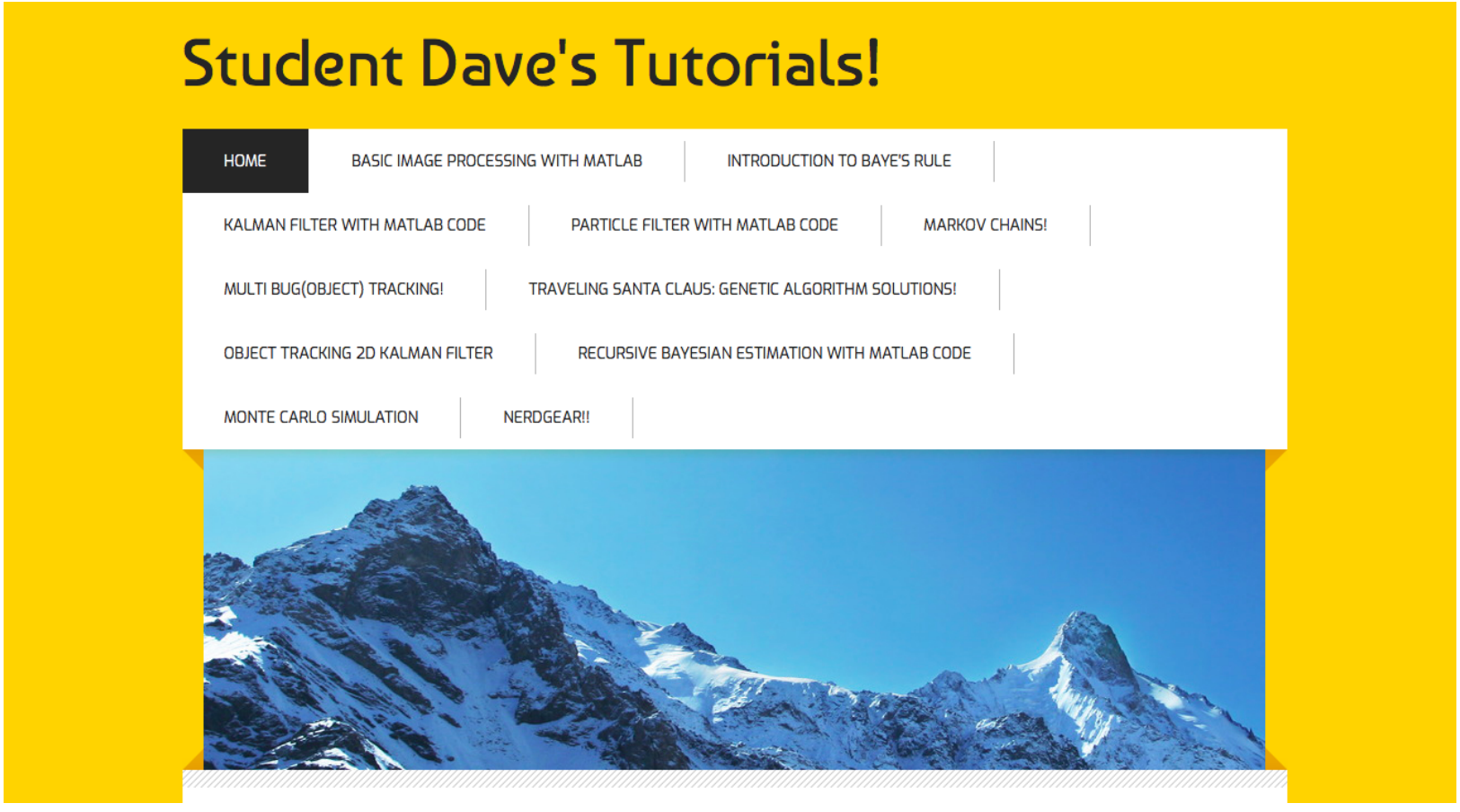
Narrowing the search space

- After some iterations, when all the ants stop moving, the distribution of intervals will have the following feature: the subintervals which contian the ants will must contain the minima; the subintervals which have no ants will impossibly have the minima
- Then, we remove all the subintervals with no ants and refine the remaining, forcing the ants search of the minima only in these subintervals
- After several iterations, the scope of searching will be smaller and smaller, and all the ants will stop nearby the minima

Genetic algorithms



The travelling salesman problem for Santa Claus



http://studentdavestutorials.weebly.com/traveling-santa-claus-genetic-algorithm-solutions.html

Genetic algorithms

As it has already been said, for large dimension problems, special artificial intelligence techniques must be employed. Genetic Algorithms (GA), developed by John Holland in the early 1970's, is one of such techniques

- 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, hopefully, to the evolution of better populations than previous populations

Genetic algorithms. What nature does (approximately)

Consider a population of rabbits and foxes in an isolated medium

- 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
- Of course, some of the slower and/or 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, some fast with fast, some smart 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

Evolution programs

Genetic Algorithms are a particular kind of procedures based on principles of evolution and hereditary. They receive the generic name of Evolution Programs.

Some Evolution Programs, are:

1. **Genetic Algorithms** (John Holland 1970's)
2. **Evolution Strategies**
3. **Evolutionary Programming**
4. **Scatter Search techniques**

All Evolution Programs:

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

Evolution programs. Main characteristics

- An **evolution program is a probabilistic iterative algorithm** which, at each iteration $t \rightarrow t+1$, **maintains a population** of individuals

$$P(t) = (x_1^t, x_2^t, ..., x_M^t) \longrightarrow P(t+1) = (x_1^{t+1}, x_2^{t+1}, ..., x_M^{t+1})$$

- The **new population**

$$P(t+1) = (x_1^{t+1}, x_2^{t+1}, ..., x_M^{t+1}),$$

  is formed at each **iteration**, $t \rightarrow t+1$, by **selecting the best fit individuals** in the select step

- Each x_i^t represents a **potential solution** to the problem, and is **implemented as some data structure**. Often, the x_i^t are called **chromosomes**

- **Each potential solution x_i^t is evaluated** to give some measure of its **fitness**

Evolution programs. Main characteristics

- Some **members of the new population undergo transformations** by means of **"genetic operators"** to form new solutions:
  - There are transformations –**mutation type**– which create new individuals by a **small change in a single individual**
  - Higher order transformations –**crossover type**– which create new individuals by **combining parts from several (two or more) individuals**. These transformations produce information exchange between different potential solutions

    For example, if the parents are represented by five-dimensional vectors

    $$\boldsymbol{x}_m^{t+1} = (a_1, b_1, c_1, d_1, e_1) \text{ and } \boldsymbol{x}_k^{t+1} = (a_2, b_2, c_2, d_2, e_2)$$

    then crossing the chromosomes would produce the offspring

    $$\boldsymbol{x}_m^{t+1} = (a_1, b_1, c_2, d_2, e_2) \text{ and } \boldsymbol{x}_k^{t+1} = (a_2, b_2, c_1, d_1, e_1)$$

- **After some (usually very large) number of generations it is hoped that the program converges**,and the best individual represents a near-optimum solution

Genetic algorithms

- **Initialization**
  - Create initial population of size M
  - Define objective (fitness) function f
  - Encode the population as chromosomes (bit strings) of length L
  - Compute fitness values of the entire population $f(\boldsymbol{x})$
  - Define termination condition, if any
  - Choose maximum number of iterations *MaxIter*
  - Set $iter = 1$
- **while** $(iter \leq MaxIter)$ **do**
  - *Selection:* select parents for reproduction
  - *Crossover:* apply crossover on parents to produce offsprings
  - *Mutation:* apply mutation on selected chromosomes (optional)
  - Compute the fitness values of the population
  - Select members for the next generation based on fitness values
  - If termination condition met exit, else continue
  - $iter = iter + 1$
- **end while**

The **chromosome with highest fitness is the optimum solution**

Genetic algorithms. Example

Assume we **search for a graph** which should satisfy some requirements

The **requirements** can be, for instance: **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 incorporates the problem requirements.** The evaluation function returns the fitness of each graph, distinguishing between better and worse individuals

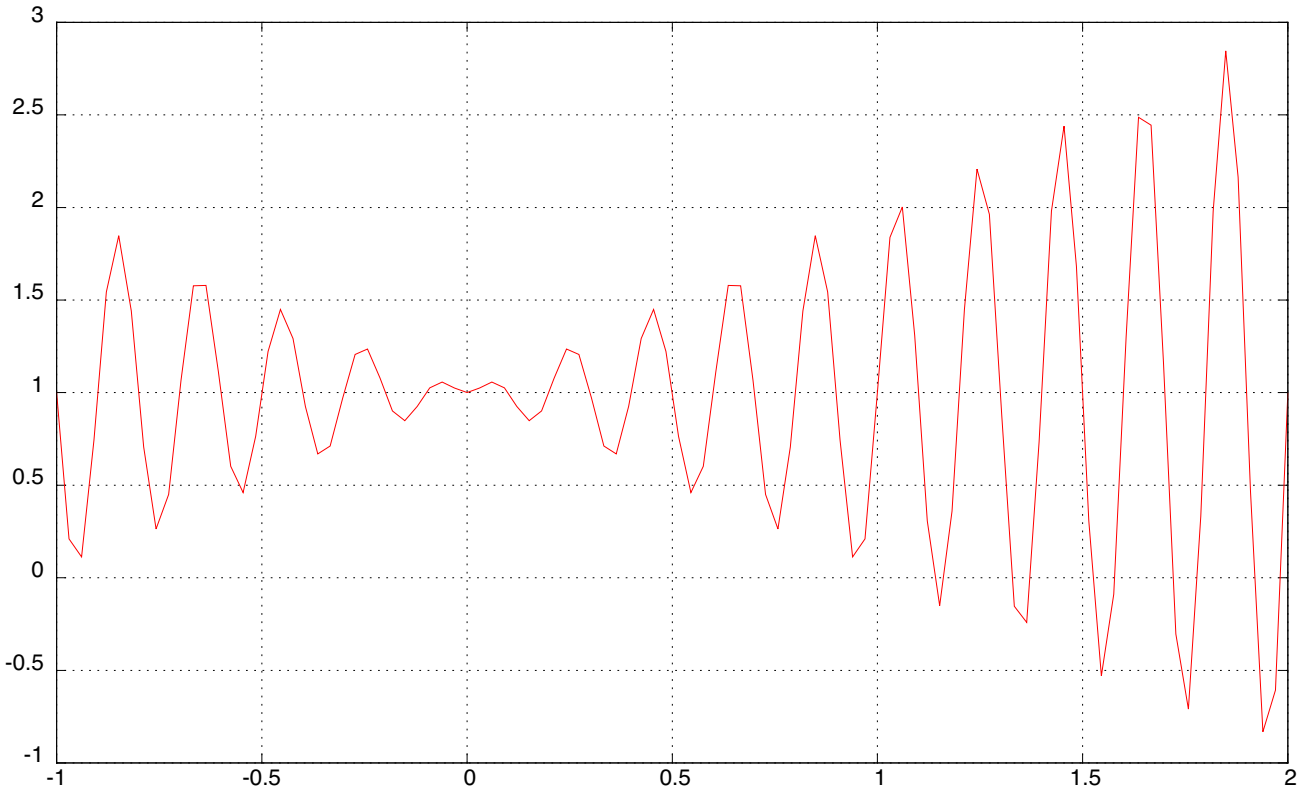
Genetic algorithms. Example

- 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

Example: optimization of a function

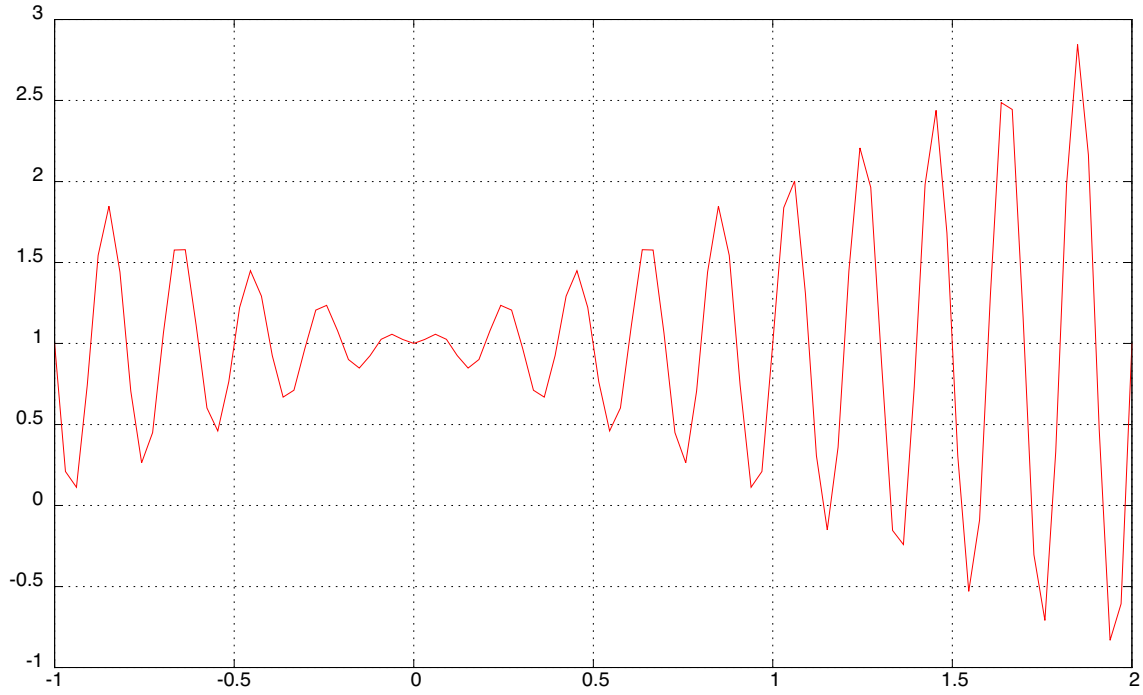
Problem: Find $x \in [-1, 2]$ that maximizes the function

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



Graph of $f(x)$

Optimization of a function



- Since the domain of the problem is $[-1, 2]$, the function

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

reaches its maximum at $x \approx 1.85$, and the value of $f(x) \approx 2.28$

- We are going 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 precision will be **six places after the decimal point** (10^{-6})
- The precision requirement implies that the range $[-1.0, 2.0]$ should be divided into, at least,

$$3 \times 1\,000\,000 \;=\; 3\,000\,000$$

  equal size ranges

- Since

$$2\,097\,152 = 2^{21} < 3\,000\,000 < 2^{22} = 4\,194\,304,$$

  this means that **22 bits are required for each chromosome (value of x)**

Optimization of a function. Representation

- The mapping from a binary string into a real number

$$(b_{21}b_{20}...b_1b_0)_2 \longrightarrow x \in [-1, 2]$$

is done in two steps

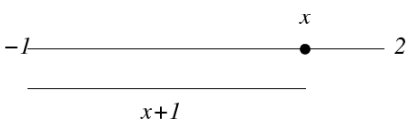
  - Convert the binary string $(b_{21}b_{20}...b_1b_0)_2$ from the base 2 to base 10

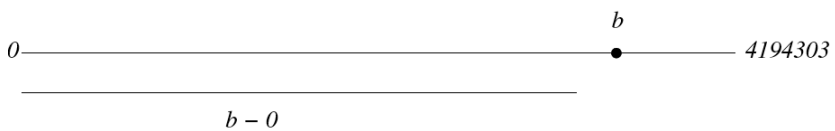
$$(b_{21}b_{20}...b_1b_0)_2 = \sum_{i=0}^{21} b_i\, 2^i = b \in [0, 2^{22} - 1]$$

  - Find a corresponding real number x in the range $[-1, 2]$

$$\begin{array}{ccc} [0, 2^{22} - 1] & \longrightarrow & [-1, 2] \\ b & \longrightarrow & x \end{array}$$

$$\frac{x+1}{3} = \frac{b}{2^{22}-1} \quad \Rightarrow \quad x = \frac{3}{2^{22}-1}b - 1$$





Optimization of a function. Representation

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

$$b = (10001011101101010001 11)_2 =$$

$$= 2^0 + 2^1 + 2^2 + 2^6 + 2^8 + 2^{10} + 2^{11} + 2^{13} + 2^{14} + 2^{15} + 2^{17} + 2^{21} = 2\,288\,967$$

and

$$x = \frac{3}{4\,194\,303} 2\,288\,967 - 1.0 = 0.637196$$

- 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 of the GA** is simple

- **Create a population of chromosomes** of a given population size
- 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 **fitness function** is the function f evaluated at the chromosomes (binary vectors b). We call it the **evaluation function**

In the example, we want to maximize the function

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

so, the **evaluation function** will be

$$\text{eval}(b) = f(x) = x \sin(10\,\pi\,x) + 1$$

where **the chromosome b (in base 2) represents the real value** x (in base 10)

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

Optimization of a function. Evaluation function

For example, the chromosomes

$$\begin{aligned} b_1 &= (10001011101101010001 11)_2 \\ b_2 &= (00000011100000000010000)_2 \\ b_3 &= (11100000001111111000101)_2 \end{aligned}$$

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:

$$\begin{aligned} \text{eval}(b_1) &= f(x_1) = 1.586345 \\ \text{eval}(b_2) &= f(x_2) = 0.078878 \\ \text{eval}(b_3) &= f(x_3) = 2.250650 \end{aligned}$$

Clearly, the chromosome b_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 the two genetic operators:
  - **mutation**
  - **crossover**
- **Mutation alters one or more genes** (positions in a chromosome) with a **probability** equal to a given **mutation rate**
- Usually the mutation rate p_m is chosen to be very low (e.g., 0.01, 0.001)

Optimization of a function. Genetic operations

- Assume that the fifth gene from the b_3 chromosome was selected for a mutation (we will see how to do it). Since the fifth gene in this chromosome is 0, it would be flipped into 1. So the chromosome

$$b_3 = (11100000001111111000101)_2$$

after this mutation would be

$$b_3 = (11101000001111111000101)_2$$

- This chromosome represents the value

$$x_3' = 1.721638 \quad \text{and} \quad f(x_3') = -0.082257$$

This means that this particular mutation resulted in a significant decrease of the value of the chromosome b_3, since $f(x_3) = 2.250650$

- On the other hand, if the 10th gene was selected for mutation in the chromosome b_3, then

$$b_3 = (11100000011111111000101)_2$$

Then

$$x_3'' = 1.630818 \quad \text{and} \quad f(x_3'') = 2.343555$$

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 b_2 and b_3
- Assume that the **crossover point** was **randomly** selected after the 5th gene

$$\begin{aligned} b_2 &= (00000\,|\,0111000000010000)_2 \\ b_3 &= (11100\,|\,0000011111000101)_2 \end{aligned}$$

The two resulting offspring are

$$\begin{aligned} b_2' &= (00000\,|\,0000011111000101)_2 \\ b_3' &= (11100\,|\,0111000000010000)_2 \end{aligned}$$

The evaluation of these two offsprings gives

$$\begin{aligned} f(x_2') &= f(-0.998113) = 0.940865 \\ f(x_3') &= f(1.666028) = 2.459245 \end{aligned}$$

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

Optimization of a function. Genetic operations

- For the **crossover** operator a **probability of crossover** p_c must be fixed
- This probability **gives the expected number of chromosomes which undergo the crossover**
- Usually, the probability of crossover p_c is chosen to be fairly high (e.g., $0.2 \sim 0.8$)

Optimization of a function. Experimental results

Consider the following simulation parameters for this problem

- population size $M = 50$
- 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

$$b_{max} = (11110011010001000001 01)_2$$

  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

Remark: In this example, we have not done any **selection process** before the mutation and the crossover operations

Optimization of a function. The k-dimesional case

- Suppose we wish to maximize a function of k variables

$$\begin{array}{rccc} f: & \mathbb{R}^k & \rightarrow & \mathbb{R} \\ & (x_1, ..., x_k) & \rightarrow & f(x_1, ..., x_k) \end{array}$$

- Suppose 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, in order to make it positive

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

Optimization of a function. The k-dimesional case

- 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 a binary string to a decimal number x_i

$$x_i = a_i + \frac{b_i - a_i}{2^{m_i} - 1} \times [(\text{binary string})_2]$$

- 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

Optimization of a function. The k-dimesional case

- To **initialize a population**, we can simply set some number M 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 chromosomes)
  - **select the new population** 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 optimal solution (eventually the global)
- Often we stop the algorithm after a fixed number of iterations depending on speed and resource criteria

The selection process

Selection of a new population

It can be done with different procedures: roulette wheel with slots, tournament selection, stochastic universal sampling,...

Roulette wheel with slots

We construct such a roulette wheel as follows

- Calculate the **fitness value, $f(b_i)$, for each chromosome** b_i, for $i = 1, ..., M$, where M is the population size
- Find the **total fitness** of the population

$$F = \sum_{i=1}^{M} f(b_i)$$

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

$$p_i = \frac{f(b_i)}{F}$$

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

$$q_i = \sum_{j=1}^{i} p_j, \quad \text{this is:} \quad q_1 = p_1,\ q_2 = p_1 + p_2,\ q_3 = p_1 + p_2 + p_3, ...$$

The roulette wheel with slots (cont.)

The **selection process**, of the the roulette wheel with slots procedure, is based on

- **Spinning the roulette wheel M times**
- **At each spinning 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 b_1
  - Otherwise, **select the** $i - th$ **chromosome** b_i $(2 \leq i \leq M)$ such that

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

this is, r **is between the two consecutive cumulative probabilities** q_{i-1} **and** q_i

The roulette wheel with slots (cont.)

With the roulette wheel with slots some chromosomes would be selected more than once. This is in accordance with the *Schema Theorem (see later): the best chromosomes get more copies, the average stay, and the worst die off*

When the selection process is finished, after M spinnings of the roulette (selections), we are ready to apply the **mutation and crossover** operators to the individuals in the new population

The selection process. Tournament selection

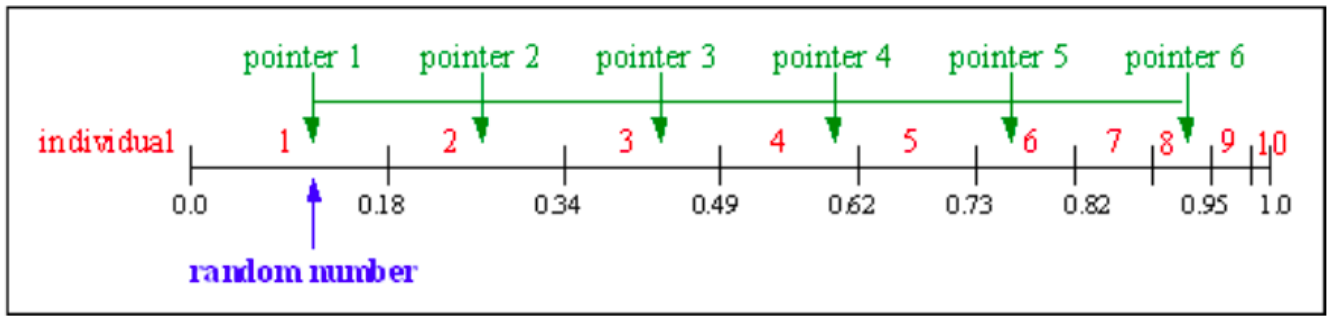
Tournament selection

- Tournament selection involves running several "tournaments" among a certain number of chromosomes (that depends on M, and usually is not too large) chosen at random from the population
- The winner of each tournament (the one with the best fitness) is selected for the new population
- Selection pressure is easily adjusted by changing the tournament size
- If the tournament size is large, weak individuals have a small chance to be selected
- The procedure is repeated until a "new" population of M individuals is generated (M tournaments).

The selection process. Stochastic universal sampling

Stochastic universal sampling

- In the stochastic universal sampling, the individuals b_i are first mapped to contiguous segments of a line, such that each individual's segment is equal in size to its fitness $f(b_i)$

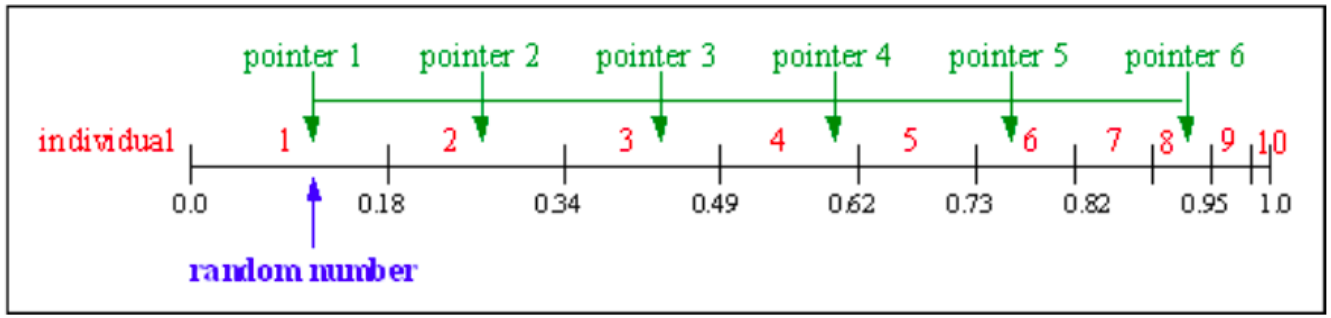


- The procedure makes a **single spin** of the roulette wheel, that **provides a starting position** (random number $r \in [0, 1]$) and the first selected individual

- 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. In this way, equally spaced pointers are placed over the line as many as there are individuals to be selected

Stochastic universal sampling (cont.)

- 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

- Stochastic universal sampling can have bad performance when a member of the population has a really large fitness in comparison with other members

The crossover operator

- 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
- The value of p_c is chosen to be fairly high ($0.2 \sim 0.8$)

The crossover operator

The crossover operator proceeds in the following way

- For each chromosome in the population
  - Generate a random number r from the range $[0, 1]$
  - If $r < p_c$ select the chromosome for crossover
- Next mate all the selected chromosomes randomly
  - For each pair of selected chromosomes we generate a random integer number, pos, in 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
  - Then each pair of selected 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 operator

- The **mutation** operator is performed on a bit-by-bit basis
- The probability of mutation p_m, gives us the expected number of mutated bits p_m. Usually, p_m is chosen to be very low (e.g., 0.001)
- Every bit of the chromosomes in the whole population has an equal chance to undergo mutation, i.e., change from 0 to 1 or vice versa
- The operator proceeds in the following way
  - For each chromosome in the current population (i.e., after selection and crossover), and for each bit within the chromosome, generate a random number r in the range $[0, 1]$
  - If $r < p_m$ mutate the bit

The iterative procedure

- 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

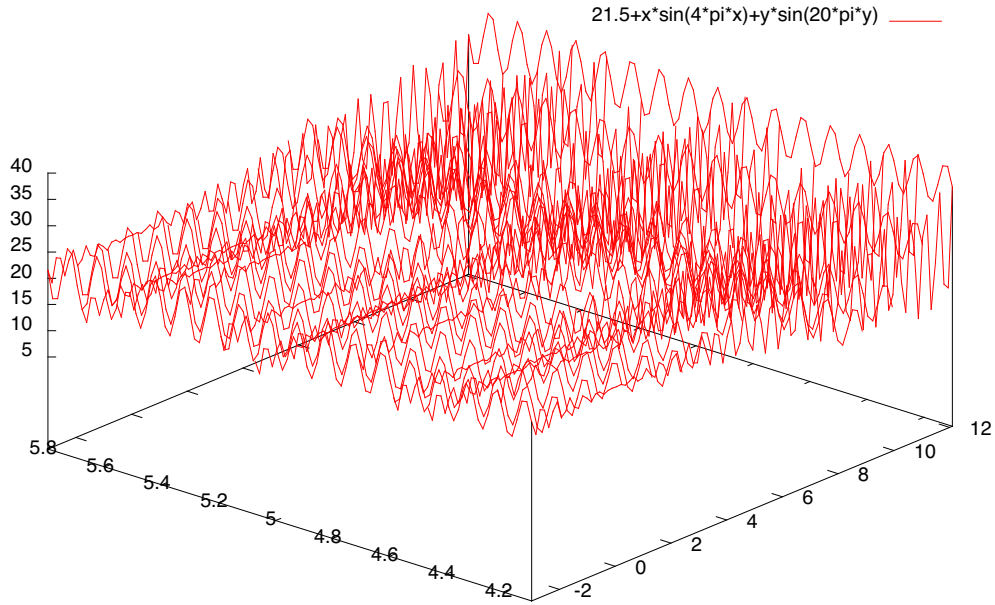
Example

We want to maximize the following function

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

with

$$(x, y) \in [-3.0, 12.11] \times [4.1, 5.8]$$



We will use a GA with a population size $M = 20$, 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", $[-3.0, 12.11]$, has length 15.1. The precision requirement implies that the x-range $[-3.0, 12.11]$ should be divided into at least 15.1×10^4 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", $[4.1, 5.8]$, has length 1.7. The precision requirement implies that the range $[4.1, 5.8]$ should be divided into at least 1.7×10^4 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

$$(0100010010110100001111110010100010)$$

- The first 18 bits

$$0100010010110100001$$

represent

$$x = -3 + \frac{12.11 - (-3.0)}{2^{18} - 1}\text{decimal}(0100010010110100001)_2 = 1.052426$$

The next 15 bits

$$111110010100010$$

represent

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

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

$$
\begin{aligned}
b_1 &= (1001101000000001111\ 111010011011111)\\
b_2 &= (1110001001001101111\ 001010100011010)\\
b_3 &= (0000100000110010000\ 001010111011101)\\
b_4 &= (1000110001011010011\ 111000001110010)\\
b_5 &= (0001110110010100111\ 010111111000101)\\
b_6 &= (0001010000100101010\ 001010111111011)\\
b_7 &= (0010001000001101011\ 111011011111011)\\
b_8 &= (1000011000011101000\ 010110101100111)\\
b_9 &= (0100000001011101000\ 010110101100111)\\
b_{10} &= (0000011110001100000\ 011010000111011)\\
b_{11} &= (0110011111101101011\ 100001101111000)\\
b_{12} &= (1101000101111011011\ 000101010000000)\\
b_{13} &= (1110111110100010000\ 110000001000110)\\
b_{14} &= (0000100110000010100\ 100111100101001)\\
b_{15} &= (1110111011011100000\ 100011111011110)\\
b_{16} &= (1100111100000111111\ 100001101001011)\\
b_{17} &= (0110101111110011111\ 010001101111101)\\
b_{18} &= (0111010000000011111\ 010011110101101)\\
b_{19} &= (0001010100111111111\ 110000110001100)\\
b_{20} &= (1011100101100111100\ 110000101111110)
\end{aligned}
$$

Example (cont.)

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

$$
\begin{array}{ll}
f(b_1) = 26.019600 & f(b_2) = 7.580015 \\
f(b_3) = 19.526329 & f(b_4) = 17.406725 \\
f(b_5) = 25.341160 & f(b_6) = 18.100417 \\
f(b_7) = 16.020812 & f(b_8) = 17.959701 \\
f(b_9) = 16.127799 & f(b_{10}) = 21.278435 \\
f(b_{11}) = 23.410669 & f(b_{12}) = 15.011619 \\
f(b_{13}) = 27.316702 & f(b_{14}) = 19.876294 \\
f(b_{15}) = 30.060205 & f(b_{16}) = 23.867227 \\
f(b_{17}) = 13.696165 & f(b_{18}) = 15.414128 \\
f(b_{19}) = 20.095903 & f(b_{20}) = 13.666916
\end{array}
$$

It is clear, that the chromosome b_{15} is the best one, and the chromosome b_2 the worst

Next, construct a roulette wheel with slots for the selection process

The total fitness of the population is

$$F = \sum_{i=1}^{20} f(b_i) = 387.776822$$

Example (cont.)

The **probability of a selection** p_i for each chromosome is

$$p_1 = f(b_1)/F = 0.067099 \qquad p_2 = f(b_2)/F = 0.019547$$

$$p_3 = f(b_3)/F = 0.050355 \qquad p_4 = f(b_4)/F = 0.044889$$

$$p_5 = f(b_5)/F = 0.065350 \qquad p_6 = f(b_6)/F = 0.046677$$

$$p_7 = f(b_7)/F = 0.041315 \qquad p_8 = f(b_8)/F = 0.046315$$

$$p_9 = f(b_9)/F = 0.041590 \qquad p_{10} = f(b_{10})/F = 0.054873$$

$$p_{11} = f(b_{11})/F = 0.060372 \qquad p_{12} = f(b_{12})/F = 0.038712$$

$$p_{13} = f(b_{13})/F = 0.070444 \qquad p_{14} = f(b_{14})/F = 0.051257$$

$$p_{15} = f(b_{15})/F = 0.077519 \qquad p_{16} = f(b_{16})/F = 0.061549$$

$$p_{17} = f(b_{17})/F = 0.035320 \qquad p_{18} = f(b_{18})/F = 0.039750$$

$$p_{19} = f(b_{19})/F = 0.051823 \qquad p_{20} = f(b_{20})/F = 0.035244$$

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

Example (cont.)

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

Now we are ready to spin the roulette wheel 20 times

Each time we select a single chromosome for a new population

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 b_{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 b_4 is selected for the new population, etc.

The new population consists of the following chromosomes...

Example (cont.)

$$
\begin{aligned}
b'_1 &= (01100111111011010110000110111100) && (b_{11})\\
b'_2 &= (10001100010110100111100000111001) && (b_4)\\
b'_3 &= (00100010000011010111011011111011) && (b_7)\\
b'_4 &= (01100111111011010110000110111100) && (b_{11})\\
b'_5 &= (00010101001111111111000011000110) && (b_{19})\\
b'_6 &= (10001100010110100111100000111001) && (b_4)\\
b'_7 &= (11101110110111000010001111101111) && (b_{15})\\
b'_8 &= (00011101100101001101011111000101) && (b_5)\\
b'_9 &= (01100111111011010110000110111100) && (b_{11})\\
b'_{10} &= (00001000001100100000101011011101) && (b_3)\\
b'_{11} &= (11101110110111000010001111101111) && (b_{15})\\
b'_{12} &= (01000000010111010001011010110011) && (b_9)\\
b'_{13} &= (00010100001001010100101011111011) && (b_6)\\
b'_{14} &= (10000110000111010001011010110011) && (b_8)\\
b'_{15} &= (10111001011001111001100010111110) && (b_{20})\\
b'_{16} &= (10011010000000111111101001101111) && (b_1)\\
b'_{17} &= (00000111100011000001101000011011) && (b_{10})\\
b'_{18} &= (11101111101000100011000000100011) && (b_{13})\\
b'_{19} &= (11101110110111000010001111101111) && (b_{15})\\
b'_{20} &= (11001111000001111110000110100101) && (b_{16})
\end{aligned}
$$

Example (cont.)

Now we are ready to apply the recombination operator, **crossover**, to the individuals in the new population (vectors b'_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 b'_2, b'_{11}, b'_{13} and $b_{18'}$ are 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., b'_2 and b'_{11}) and the next two (i.e., b'_{13} and b'_{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

$$\begin{aligned} b'_2 &= (100011000101101001111000001110010) \\ b'_{11} &= (111011101011100001000111111011110) \end{aligned}$$

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

$$\begin{aligned} b''_2 &= (111011101011010011110000011100 10) \\ b''_{11} &= (100011000101110000100011111011110) \end{aligned}$$

Example (cont.)

The second pair of chromosomes is

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

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

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

The current population is now...

Example (cont.)

$$
\begin{aligned}
b'_1 &= (01100111111011010110000110111000)\\
b''_2 &= (11101110110110100111100000111001 0)\\
b'_3 &= (00100010000011010111011011111011)\\
b'_4 &= (01100111111011010110000110111000)\\
b'_5 &= (00010101001111111110000110001100)\\
b'_6 &= (10001100010110100111100000111001 0)\\
b'_7 &= (11101110110111000100011111011110)\\
b'_8 &= (00011101100101001101011111000101)\\
b'_9 &= (01100111111011010110000110111000)\\
b'_{10} &= (00001000001100100000101011011101)\\
b''_{11} &= (10001100010111000010001111011110)\\
b'_{12} &= (01000000101110100010110101100111)\\
b'_{13} &= (11101111101000100011010111111011)\\
b'_{14} &= (10000110000111010001011010110011 1)\\
b'_{15} &= (10111001011001111001100010111111 0)\\
b'_{16} &= (10011010000000111111101001101111 1)\\
b'_{17} &= (00000111100011000001101000011101 1)\\
b''_{18} &= (00010100001001010100000000100011 0)\\
b'_{19} &= (11101110110111000100011111011110)\\
b'_{20} &= (11001111000001111110000110100101 1)
\end{aligned}
$$

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 $m \times M = 33 \times 20 = 660$ bits in the whole population, so we expect 6.6 mutations in each 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, only 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

$$
\begin{aligned}
b_1 &= (01100111111011010110000110111000)\\
b_2 &= (11101110110110100111100000111001 0)\\
b_3 &= (00100010000011010111011011111011)\\
b_4 &= (01100111111011010110000110111000)\\
b_5 &= (00010101001111111110000110001100)\\
b_6 &= (10001100010110100111100000111001 0)\\
b_7 &= (11101110110111000100011111011110)\\
b_8 &= (00011101100101001101011111000101)\\
b_9 &= (01100111111011010110000110111000)\\
b_{10} &= (00001000001100100000101011011101)\\
b_{11} &= (10001100010111000010 0011111011110)\\
b_{12} &= (01000000010111010001011010110011 1)\\
b_{13} &= (00010100001001010100101011111110 1)\\
b_{14} &= (10000110000111010001011010110011 1)\\
b_{15} &= (10111001011001111001100010111110)\\
b_{16} &= (10011010000000111111101001101111 1)\\
b_{17} &= (00000111100011000001101000011101 1)\\
b_{18} &= (00010100001001010100000001000110)\\
b_{19} &= (11101110110111000100011111011110)\\
b_{20} &= (11001111000001111110000110100101 1)
\end{aligned}
$$

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

$f(b_1) = 23.410669$	$f(b_2) = 18.201083$
$f(b_3) = 16.020812$	$f(b_4) = 23.1412613$
$f(b_5) = 20.095903$	$f(b_6) = 17.406725$
$f(b_7) = 30.060205$	$f(b_8) = 25.341160$
$f(b_9) = 23.410669$	$f(b_{10}) = 19.526329$
$f(b_{11}) = 33.351874$	$f(b_{12}) = 16.127799$
$f(b_{13}) = 22.692462$	$f(b_{14}) = 17.959701$
$f(b_{15}) = 13.666916$	$f(b_{16}) = 26.019600$
$f(b_{17}) = 21.278435$	$f(b_{18}) = 27.591064$
$f(b_{19}) = 27.608441$	$f(b_{20}) = 23.867227$

The total fitness of the new population F is 447.049688, which is much higher than total fitness of the previous population, 387.776822

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

Example (cont.)

After 1000 generations the fitness values of the population are

$$
\begin{array}{ll}
f(b_1) = 30.298543 & f(b_2) = 26.869724 \\
f(b_3) = 30.316575 & f(b_4) = 31.933120 \\
f(b_5) = 30.316575 & f(b_6) = 34.356125 \\
f(b_7) = 35.458636 & f(b_8) = 23.309078 \\
f(b_9) = 34.393820 & f(b_{10}) = 30.316575 \\
f(b_{11}) = 35.477938 & f(b_{12}) = 35.456066 \\
f(b_{13}) = 30.316575 & f(b_{14}) = 32.932098 \\
f(b_{15}) = 30.746768 & f(b_{16}) = 34.359545 \\
f(b_{17}) = 32.932098 & f(b_{18}) = 32.956664 \\
f(b_{19}) = 19.669670 & f(b_{20}) = 32.956664
\end{array}
$$

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 (cpu) time has elapsed. Note: If the maximum number of generations 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 generations has been reached before the specified number of generations with no changes has been reached, the process will 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 cpu time in seconds equal to Stall time limit

Exercise

Exercise 9. To be delivered before 24-XI-2021 (`Ex09-YourSurname.pdf`)

Use a genetic algorithm to solve the example of page 84 with population size $M = 50$*:*

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

with

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

The Fundamental Theorem of Genetic Algorithms

The **Fundamental Theorem of Genetic Algorithms**, also called the Schema theorem, says, in short, that low-order schema with above-average fitness increase exponentially in successive generations

Definitions:

- A schema (building block) is a template that identifies a **subset of strings with similarities** at certain string positions

  Example: | 1 | x | 1 | 0 | x | 1 | in a chromosome (long strings of 0's and 1's). This schema has length 6 with 1's at positions 1, 3 and 6, and a 0 at position 4. The "x' is a wildcard symbol, which means that positions 2 and 5 can have a value of either 1 or 0

- The order of a schema is defined as the **number of fixed positions in the template**. In the example the order is 4

- The defining length is the **distance between the first and last specific positions**. In the example the defining length is $6 - 1 = 5$

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

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 average fitness of the schema H
- $o(H)$ be the order of the schema
- l be length of the strings (chromosomes)
- $\delta(H)$ be the defining length of the schema H
- $\overline{f}$ be the average fitness of the current population
- p_c be the crossover probability
- p_m be the mutation probability

Then, the espectation of $m(H, t+1)$ is

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

The above fundamental theorem states that schemas with mean fitness $f(H)$ greater than the population average fitness $\left(m(H, t)\frac{f(H)}{\overline{f}}\right)$, short defining length $\left(1 - p_c\frac{\delta(H)}{1-l}\right)$, and lower order $\left((1-p_m)^{o(H)}\right)$ are more likely to survive

Chromosome representations. The Traveling Salesman Problem (TSP)

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 cities, 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 ordered cities, this is 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 $\{1, 2, ..., n\}$

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 $\{1, 2, ..., n\}$
- 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

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