COMENIUS UNIVERSITY IN BRATISLAVA
FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS
DEPARTMENT OF APPLIED INFORMATICS

Introduction to Artificial Intelligence

Exercise 4 - Genetic Algorithm: choosing parents, crossover and mutation

October 12, 2022

GENETIC ALGORITHM

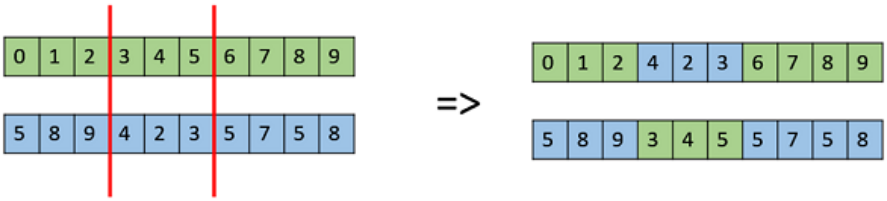
MAIN COMPONENTS OF GENETIC ALGORITHM:

GA is an iterative algorithm of local search, in each iteration (generation) the solution gets improved. Each iteration consists of following stepsy:

- Rating of each individual from population by evaluating its fitness.
- Choosing parents - pairs for crossover. Each pair of parents generates (usually two) children by crossover.
- Mutation - chosen individuals are with small probability slightly modified.
- Choosing next generation: combination of current population (parents) and children.

Each of these steps can be realized in multiple ways. We will implement this:

- Rate the individuals.
- Choose parents as the $N/2$ best individuals. Pairs (for crossover) from the set of parents can be chosen randomly, or by some criterion.
- Each pair of parents generates two children by k-point crossover - sequences of both parents are crossed at **exactly *k*** random places, generating two children. For $k = 2$:



Children undergo mutation - each component of a given individual is changed with a small probability p. Implement two types of mutation:

- Bit mutation: if an individual is represented as a list of bits (for example $[0,1,1,0,1]$) mutation means flipping a bit from 0 to 1 or vice versa.

- Number mutation: if an individual is represented as a list of real numbers (for example $[0.3, 1.0, 0.0, 0.8, 0.6]$), mutation means adding a random number from $\mathcal{N}(0, \sigma^2)$ (normal distribution).

New generation consists of:

- Best $N/2$ individuals from the current population (parents).
- $N/2$ children, after undergoing mutation.

Program:
In *problems.py* there are 3 pre-defined problems to solve with GA: string of ones, drawing a smiley face and painting a masterpiece. These three classes inherit from *GenAlgProblem*, and functions *fitness(x)* a *mutation(x)* are already implemented.

genetic.py contains the main class *GenAlgProblem*, which you need to finish:

- *crossover(x, y, k)* (0.2p) - given parents x and y, do k-point crossover, and return two lists - children.
- *boolean_mutation(x, prob)* (0.15p) - each bit of individual x is changed with probability $prob$.
- *number_mutation(x, prob)* (0.15p) - each number of individual x is changed with probability $prob$ by adding a random number.
- *solve(max_generations, goal_fitness)* (0.5p) - runs the GA, using aforementioned functions. Algorithm stops after $max\_generations$ generácií, or after obtaining an individual with fitness greater or equal to $goal\_fitness$.
  In your implementation, you can assume that the size of population is divisible by 4.

A few details:

1. Population *self.population* is a list of individuals, each individual is a list of numbers.
2. The class *GenAlgProblem* has defined variables *self.n_crossover* and *self.mutation_prob*, use them in *solve()* instead of constants.
3. Function *solve()* should return the best individual.
4. Generating of the initial population is already implemented.
5. For testing function *crossover(x, y, k)*, there is a short code at the end of *genetic.py*.

Assignment (1p): Implement the functions to solve all three problems. In the third problem, resulting fitness should be at least 0.75. During debugging/testing you can try changing parameters (size of population, number of crosses in crossover, mutation probability, sigma in number mutation) to see how they affect the performance.
Bonus (0.5p): Implement two more sophisticated ways of choosing parents: roulette and tournament (you can find some literature online, for example HERE). Write shortly (directly in code or in readme.txt) what you think about all the implemented methods, which one is the best or in what situations it is best to use each of them.