NAT Tutorial 1

1. The knapsack (or: rucksack) problem is as follows: given a set of weights W, and a target weight T, find a subset of W whose sum is as close to T as possible.

Example: $W = \{5, 8, 10, 23, 27, 31, 37, 41\}$, $T = 82$

a) Solve the instance of the knapsack problem given above.

Answer: 82 = 41+31+10 is one solution. Another is 41+23+10+8. Any other solutions?

b) Consider solving the knapsack problem using the canonical GA. How can a solution be encoded as a chromosome?

Answer: If the weights set is of size W, then each bit in a chromosome encodes whether item $w[i]$ is present or not.

c) What fitness function can be used for the knapsack problem, so that better solutions have higher fitness?

Answer: If D is the total sum of the candidate solution, $f(D) = 1/(1+|T-D|)$. Think of possible alternative functions if possible, e.g. linear tent-shaped function, $T-|T-D|$. See also question e)

d) Given your answer to question b, what selection methods would be appropriate?

Answer: Rank or Tournament. Fitness-proportionate may give too much emphasis to strong solutions, which may not be important in the present example, but the population may be dominated by individuals that contain the largest item(s), although the optimal solution may consist of many small ones.

e) Consider also the case that the total weight of the subset must not exceed T. Would you need to change your approach?

Answer: Whether solutions with D>T are acceptable depends on the context, but it would often be reasonable to say that D>T is not admissible. In this case the fitness should be $(T-D)\Theta(T-D)$, i.e. $f(D)=T-D$ if $T \geq D$ and $f(D)=0$ otherwise (Θ is the Heaviside step function). Note that this fitness function is not deceptive, although it may help to allow initially violations of admissibility in the sense that a symmetric fitness is used, while the non-exceedance enforce only in later generations.

2) Assume you have a lot of data points that seem to fall into clusters, e.g. the 2D position of mushrooms in a forest. Instead of applying the k-means algorithm directly, you decide to use GA to get the centre positions of the clusters. How might you use a canonical GA to solve this, and what are the problems you might run into, particularly regarding the representation?

Answer: k-means assigns k "centroid" vectors to a data distribution inducing a Voronoi tessellation over which a mean distance can be defined by the average over the distances of the data points in one Voronoi cell to the centroid. An individual in the GA represents such an assignment and the average error for this assignment can be used for determining the fitness. Mutation should respect the importance of the bits that encode the centroids, but is essentially uncritical. If the algorithm is already close to a good solution then cross-over might lead to strange effects because the order of the centroids might be different for each individual.

A further question would be whether it makes sense to evolve not a population of assignments of k vectors to the data, but a population of k individual each of which represent a single centroid that competes with other centroids for data.

Answer: This idea underlies in a sense the standard *k*-means, but is beyond any standard GA.

3. Run through a simple GA (again without a computer!), applying fitness proportionate selection and single-point crossover. It could be the "maximize $f(x) = x^2$ (x-squared)"-problem from the notes. Set up a population of individuals by tossing a coin to get the initial chromosomes and use coin-tossing wherever you need to generate random numbers. Note how the average fitness, sum of individual fitness values, and maximum fitness change over the generations. You may need a calculator for this so bring a laptop or a mobile phone (or even a calculator if they still exist), but before you start with the experiments, consider what range for x, what encoding, what population size, and what parameters are reasonable here, and make a guess how many generations will be needed until the optimal solution is found.

   Answer: **"**Experience is simply the name we give our mistakes." (Oscar Wilde)

   How does the required number of generations depend on the population size? For the "all-ones" problem an inverse linear scaling is expected, but in general scaling can be slower, e.g. if the population loses diversity, or (possibly) faster, if building blocks (that account for a good part of the optimal fitness) can be discovered in parallel and then be combined.

   Among possible fitnesses, consider different exponents a, i.e. $f(x) = |x|^a$. What values for a would be useful for a good scaling? For $a<1$ the good values would be closer together in fitness, for $a>>1$ the fitness would be more distinct close to the optimum. The answer will depend on the operators, e.g. how much diversity is introduced and also whether selection is done according to rank or fitness-proportional.

   Would it be a good idea to use this approach in real problem? (sure, but obviously using a computer, except possibly for the fitness determination).

   In the high-dimensional case the problem does not get much more complicated: It would be linear in n or better, because the population-based algorithm can combine building blocks. The dependence on a would become more and more dramatic for higher n.

4. **Computer exercise** (adapted from Mitchell): Implement a simple GA with fitness-proportionate selection, roulette-wheel sampling, N=100, p_c=0.7, p_m=0.001. As fitness, use the integer value that is obtained when considering the genome of an individual as a binary number.

   a) How does the number of generations needed to find the optimum depend on the size of the genome? This is just an exercise, not an assignment, so just try a few sizes and record the result.

   b) Compare your result with a hill-climbing algorithm on the same problem.

   c) The travelling salesperson problem asks to find the shortest path through a set of N cities given the pairwise distances. Create randomly the (x,y) positions of 20 - 50 cities, determine their distances, and then mutate (how?) a string representing the tour and evolve a tour that leads back to the starting city with the shortest distance.

   Answer: Check how your program, tests, parameters, problems etc. differ from solutions by other students your group. Check if you understand the results that you have obtained (this is not always possible, but for the simple problem considered here not too difficult.

   There is no program for hill-climbing included: The idea is that because hill climbing finds the optimal solution in D steps, where D is the dimension of the search space, no computing is necessary. Note that hill-climbing also approaches the optimum optimally fast, because it checks by which mutation it achieves the largest progress. In other problems this may not work, though.

   For the GA solution, it seems from the numerics that there also a linear increase, although about 20 times slower than with hill-climbing, however already at D>50, the numerics

becomes a problem, the fitness differences are so strong that the algorithm is more greedy and gets a bit faster. Note that this is with elitism, while without elitism the results will be qualitatively similar, but slower.

Similarly, for c) knapsack problem it is important to realise that for large N the algorithm struggles to find the global optimum. It should of course be considered that we may not know the optimal solution (there are algorithms for this problem more powerful than GA!)

5. Recall what you know about natural evolution, DNA, genomes, natural selection etc. What features of biological evolution are reflected in GAs and how could GAs be improved by including additional features into the algorithm design?

   Answer: Genetic algorithms are only rough sketch of natural evolution. Most importantly, the idea of a fitness function that can be evaluated for a single individual is not meaningful in nature where the fitness of a species always depends on the fitness of other individual and other species. The logic is therefore different: In GA we use the fitness function to decide which individuals are selected for the next generation, whereas in biology fitness can be defined indirectly by the ability of an individual to produce offspring. There are many interesting observations some of which we will discuss later in the course such as

   - The possibility to improve fitness by learning and adaptation (*Baldwin effect*), which is in a sense what we try to achieve by including a hill-climbing stage into the process.
   - The occurrence of "junk"-DNA. Usually we make sure to use an efficient representation, but in GP we will mention also cases where the presence of unused parts of code in evolving programs is considered to be beneficial for the evolution.
   - The DNA coding principles: DNA codes directly amino acids, and amino acids that can have a similar function are represented by similar codes. Such considerations will usually be taken by the user of an MHO algorithm when encoding the problem to make sure that the fitness function is smooth or even monotonous over the hypercube that represents the genes. A similar principle is useful in encoding the problem in GAs: Related properties in the problem space should be encoded by nearby section of the genome.
   - Neutral mutations: Biological mutations have often no effect for the individual, which can be a result of stabilising mechanisms or error tolerant coding, but it is also possible that these mutations are occurring to produce a maximal width of the genome of the population filling thus the ecological niche as much as possible. In GA and other MHO algorithms this would correspond to methods the increase the diversity with minimal effects on fitness.

   Note that this list can be extended by many other points. This task was meant as an exercise in finding biological inspiration for design and further improvement of MHO algorithms, not in itself as learning material.