3 Intelligent Systems Lab Assignment — Evolutionary Algorithms

In this lab you will implement key components of an evolutionary algorithm for your choice out of two standard AI problems: the knapsack problem and the traveling salesman problem. You can/should download the starting code from Canvas, which consists of a single `Main.java` file that implements a generic evolutionary algorithm, but has a few components missing. There should be sufficient comments to help you understand the provided code and where you should add your contributions.

If you want, you can add additional techniques such as elitism or additional performance monitoring such as a histogram of the fitness distribution to monitor (the lack of) population diversity.

However, you *must* design and develop:

- a suitable genotype that encodes candidate solutions,
- a suitable crossover and mutation strategy,
- a selection strategy that constructs a breeding pool of candidate solutions, and
- a fitness function that fits your chosen problem.

PLEASE DON'T STEAL A SOLUTION FROM THE INTERNET AND TURN IT IN AS YOUR OWN WORK. There are many solutions for genetic algorithms to be found, but implementing a solution for the first problem is *not that hard* and will help you much more towards passing the exam. And I promise you, you will feel much better for the rest of your life by not cheating.[1]

You can choose one of two problems to solve: The Knapsack problem and the Traveling Salesman problem. Yes, there is a complexity gap between the two. Yes, I know you did the knapsack problem in the first year. EITHER pick the easy route but see if you can do a better job given your additional training, experience and months of being alive. Also, see if you can apply what we saw in class to make your EA converge faster. OR, be brave and pick the traveling salesman problem instead.

Knapsack

From Wikipedia:

> The knapsack problem or rucksack problem is a problem in combinatorial optimisation: Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items.

You can limit your experiments to the 0-1 knapsack problem in which there is only 1 copy of each item to be packed. To generate a knapsack problem of n items, set the usefulness or reward of the packages to 1,2,3,4,... and generate random weights for the packages between 1 and 10. You can also play with the max weight allowance to vary the difficulty of the problem.

Traveling Salesman

From Wikipedia:

> The travelling salesman problem (TSP) asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?" It is an NP-hard problem in combinatorial optimization, important in operations research and theoretical computer science.

[1] Also, please be aware that I am a 20th level Jedi-Ninja when it comes to Googling and a certified expert at sniffing out "currying" (look it up) and refactorisation. And if I DO find out you submitted somebody else's work, be aware that I have a very specific set of skills and that I will use those to hunt you down and tickle you until you barf! Also, you will be reported to the Board of "Exterminators".

To be able to judge your solutions quickly, you can build TSPs or varying size in which the cities to be visited all lie equidistantly on a unit circle. This makes it very easy to check how close you are to an optimal solution. As with the knapsack problem, start small and increase the size of the problem until your systems starts to break. The distance between 2 points on a unit circle that are separated by an angle γ is $\sqrt{2 - 2cos(\gamma)}$ (i.i.r.c.).

This problem might be a bit more challenging to translate into a good genotype and to define crossover and mutation operators for. Besides being creative, you can read the paper I attached, but also try Googling TSP and crossover operators. But, if you do give in and use a solution from literature, please give them credit and reference where you got it from. If you don't it could be considered plagiarism, so avoid at all cost. (Especially when there is no cost for avoidance.)

Handing in

Please upload your code and *short* report (unzipped and in PDF format only) through the Student Portal to pass this learning goal.

Pay attention, the goal is not to reiterate how evolutionary programming work to me, as I know this already. Instead inform me about the design choices you made and the motivation for them and maybe include a view on the influence of the different parameters as shaped by the experiments you ran. Each of the two tasks has a specific problem for an evolutionary algorithm, so you should have something to motivate your choices no matter which problem you chose.