2 Intelligent Systems Lab Assignment — Evolutionary Computation

In this lab you will implement genetic programming solutions for two standard AI problems: the knapsack problem and the traveling salesman problem.[1] Since you will both have to develop a generic genetic algorithm and application specific operators and fitness evaluation, you will have two sessions of two hours to work on this in class. We suggest that you try to finish the generic code and at least get a first try on the knapsack problem in the first two hour session.

Genetic Algorithm

Design and build a generic genetic algorithm in which it is relatively easy to use different genotype representations, to alter selection, crossover and mutation methods and to vary parameters such as the size of the population, the probability for crossover and mutation, the elitism proportion, the number of generations, etc.

You should aim at designing your program in such a way that it gives you feedback about what is going on. E.g. as discussed in class, you could have your programme output a histogram of the fitness distribution to monitor (the lack of) population diversity. Of course, at the end, it should report both the fitness and structure of the best found result.

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 one is not that hard and will help you much more for passing the exam and I promise you, you will feel much better for the rest of your life by not cheating.[2]

For the two problems below, make sure you try (and report) on a number of different settings for all of the parameters of your algorithm. Yes, I left the knapsack problem in ... I expect you will probably build a better implementation than the one you made in the first year of the bachelor anyway ...

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.

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. As with the knapsack problem, start small and increase the size of the problem

[1] I don't care which language you use, but if you ask me, you will have to do it in Prolog.

[2] 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".

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, even if you're not turning in a report. If you don't it could be considered plagiarism.

Handing in

Those of you who choose to earn their daily work points through this lab assignment: please upload your code and short report (in PDF format only) through the Student Portal by the end of Friday March 6^{th} if you want to get credit for your work. The report should include a description of your genetic algorithm, and the experimental results you obtained. This is a long time from now because of the Carnival holiday, but please consider the advice to turn in it much much earlier.

Pay attention, the goal should not be to reiterate how evolutionary programming work to me, as I know this, but to inform me about the design choices you made and the motivation for them and a view of the influence of the different parameters as shaped by the experiments you ran.

For those of you who don't want to get graded, please upload your code only, if necessary augmented with a brief reference to the material you used. This is the required minimum you should submit, but more elaborate README's are appreciated.