iscte TECNOLOGIAS E ARQUITETURA

Introduction to Machine Learning

Evolutionary Algorithms Exercise

These exercises can be solved in any programming language of your choice. It is assumed you are proficient with programming. It will be helpful if the language has a library to plot graphics.

Questions: https://join.slack.com/t/introductiont-qjc9426/signup (only @iscte-iul.pt users allowed) - Remember that exercises are for evaluation so don't hint about your solutions on public channels.

1. Create a function that produces a random bit pattern of a specific size (the bit pattern may be coded as a String of 0s and 1s)

2. Create a function that will generate random patterns and measure how many attempts (and how much time) does it take to generate the "correct" bit pattern. Make two graphics on the evolution of attempts / time vs the number of bits in the pattern (8, 16, 32, ...). Record also the average run-times per pattern-size. Each "point" in the graph should be a box-plot based on the results of 30 trials. Use a fixed set of seeds to be able to reproduce the experiments.

3. Create an evaluation function that measures the proximity to a pattern. Can this information be used to improve the search-times? Demonstrate how.

4. Create another evaluation function that favors some patterns over others according to a rule of your choice. For example, the system is designed to favor patterns with the greatest number of 1 bits, the evaluation function could simply be a count of the 1 bits, the best would be 11111111 ... (you cannot use this rule, you must make one up yourself). Preferably the chosen rule should be such that:
   - the evaluation function has maximum score for a small number of bit patterns or even just one
   - the score should provide a measure of "how good the pattern is" according to the chosen rule
   - the maximum patterns are easily identifiable by looking at the bit-pattern

   Use the same algorithm designed for 3 (adapted to reach a maximum instead of a minimum) and register the average number of steps in 30 runs for each pattern-size. Record also average run-times.

5. Create a function to mutate (flip one bit) a given pattern. Use this function in a cycle where the change is accepted only if it generates a better solution. Stop when you cannot find a better solution

after 1000 mutations. Does it always converge to the best solution (even f)? If not, can you understand if there is one mutation of the last sequence that could result in a better pattern?

6. Generate a random set of patterns (a population, for example, 100 patterns) and evaluate each of the patterns. Select the best 30% and, based on these best 30%, generate (by mutation) a new set of 100, repeat the processes until the best evaluation stagnates. Compare fairly the search methods tested so far, including their run-times.

7. Try to use also a method of generating a binary pattern that copies parts of two "parent" patterns (crossover). Compare the results with the previous ones.

8. Explain the changes that would be necessary in the evaluation function, mutation and crossover to deal with a similar problem where the size of the target bit pattern would be unkown.

9. Explain the changes that would be necessary in the evaluation function, mutation and crossover to deal with a similar problem where the pattern would be of decimal numbers and not binary.