Project 2 IT3708:
Programming an Evolutionary Algorithm (EA)

Purpose: Implementing a basic EA and using it to first solve two smaller problems and then later find surprising sequences.

1 A Basic EA

See the slides and lecture notes on It's Learning for background material. In the programming language of your choice, you should implement an Evolutionary Algorithm.

Your EA must include the following aspects:

1. A **population** of potential solutions (i.e. individuals) all represented in some low-level genotypic form such as binary vectors.
2. A **developmental method** for converting the genotypes into phenotypes. For this general EA, a simple routine for converting a binary genotype into a list of integers will suffice. This can be extended for future homework modules.
3. A **fitness evaluation** method that can be applied to all phenotypes. You will want to make this a very modular component of your EA such that a wide variety of fitness functions can be experimented with
4. All 3 of the basic protocols for **adult selection** described in the ea-appendices.pdf chapter of the lecture notes. (Full, over-production and mixing)
5. A set (of at least 4) mechanisms (also described in ea-appendices.pdf) for **parent/mate selection**. These 4 must include fitness-proportionate, sigma-scaling, and tournament selection.
6. The **genetic operators** of mutation and crossover. For this project, you only need to define them for binary genomes (i.e. bit vectors). But your system should be able to handle different representations for later problems.
7. A basic **evolutionary loop** for running the EA through many generations of evolution
8. A **logging routine** that shows various data while the EA is running. This can be as simple as printing to stdout/console. For each generation, this data should be logged: generation number, best fitness, average fitness, standard deviation (SD) of fitness, phenotype of the best individual.
9. A **plotting routine** that gives a visualization of an EA run. The plot should show how the data from the logging changes from generation to generation: Generations on the X axis, and then plot the best, average and SD of the fitness. For this, you can either use an existing plotting library for your language, or just copy the logging data to a spreadsheet program like Excel and create the plots there.

Aside from the plotting routine, all of the above aspects must be coded from scratch. Your program should be highly parameterizable, so that factors such as which problem to solve, population size, number of generations, crossover and mutation rates, which parent selection to use etc. can be specified by the user at or just before run-time. You should NOT need to recompile or reload your system merely to run with a different setting of any of these parameters: they should be inputs of some form, whether from the command line, an input script, or a GUI. Recompiling for these trivial changes is grounds for point loss during a demo.

Your EA should be general enough to be reused for different problems with different fitness functions, representations etc. without needing changes to the core parts.

Different problems should all be handled by the same basic classes, although you will often need special subclasses of the genotypes and phenotypes for problem-specific representations. The genetic operators for these subclasses will often be different than those for the generic genotype class. However, many problems can reuse a) the binary genome, b) the mutation and crossover operators for binary genomes, and c) the conversion routines from binary segments to integers

2 The One-Max Problem

This is a search problem that is trivial for a human to solve but slightly more difficult for an EA. The goal is simply to find a bit vector (of some pre-determined length) containing all 1's. So for the 20-bit OneMax problem, the goal is to find the 20-bit vector containing all 1's. Since an EA initially creates genotypes at random and then uses stochastic processes (selection, crossover and mutation) to generate succeeding generations of genotypes, the EA cannot simply generate an all-ones vector to solve the problem. It must generate random vectors for the initial population and then assign them fitness based on their proximity to the goal (i.e. the all-ones vector). Those with more ones will receive higher fitness and will reproduce more often, so the total number of 1's in the population will gradually increase until an individual with all 1's emerges. So, for example, the fitness function could just count the proportion of 1's in the genotype/phenotype (they are essentially the same thing for this simple problem) and assign that fraction as the fitness.

On a 20- or 30-bit One-Max problem, an EA with a population size of 20 might take 50 or 100 generations to find the solution, but it can normally find it. If not, the EA probably has a bug. Make sure that your EA can solve the One-Max problem of size 40 (bits) before moving on to other problems.

3 The LOLZ Prefix Problem

The LOLZ Prefix problem is to maximize the number of leading ones (LO) or leading zeros (LZ) in a bit string. The fitness is the number of consecutive similar bits from the beginning of the string, with the twist that the score for having leading zeros is capped at some threshold value z. For the 6-bit problem, with z = 4, this is some examples and their fitness value: [111111]=6, [000000]=4, [110101]=2, [001011]=2.

4 Surprising Sequences

Surprising sequences are those that are completely free of repeating patterns. Defined formally:

> A sequence is surprising if and only if, for every pair of symbols, A and B, and any distance d, there is at most ONE instance in the sequence of AX_dB, where X_d is any subsequence of length d.

or not as formally:

> A string over an alphabet is surprising if there are no two symbols A and B of the alphabet such that there are two pairs of occurrences of these symbols where A precedes B by the same distance.

Note that **order matters** in assessing patterns, so AX_dB is not equal to BX_dA. Also, the criteria include the case where A=B.

As a simple example, the sequence ABCCBA is surprising, but AABCC is not, since AX_2C occurs twice. Similarly, ABBACCA is not surprising due to 2 occurrences of AX_2A.

For this assignment, we will consider two types of surprising sequences:

1. **Globally** surprising sequences are those defined above: all values of distance d must be considered.
2. **Locally** surprising sequences are those in which there are no repeat occurrences of AB (i.e. AX_0B) for any symbols A and B. That is, the only distance of relevance is d = 0.

Thus, AABCC is not globally surprising, but it is locally surprising. The same goes for ABBACCA. A sequence that is not locally surprising is ABCBC due to the repeat of BC.

Your EA should search for the longest possible surprising sequence that can be constructed from symbol sets of different lengths. For instance, when there are only 2 symbols, then the longest globally surprising sequence is ABBA (or BAAB or AABA, etc.), while the largest locally-surprising sequence is ABBAA (or BAABB or AABBA, etc.)

So, for each size (S) of symbol set, you should run your algorithm many times with different lengths (L) to see if it can construct a sequence of length L from the symbols. If it can, you can increase L and see if you can find an even longer one. You should handle S up to 40, for both locally- and globally-surprising sequences.

To give you an idea of the size range: In general, the longest L for a given S is less than 3S. For S = 10, the longest globally-surprising sequence is of L = 26. The longest locally-surprising sequence for S=10 is above 100. Don't expect your EA to find the longest possible, at least not consistently. For a given S and L, it is wise to run your EA between 5 to 10 times to see if it can find a solution of length L.

How you represent this is in your program is up to you, but when displaying a solution you should follow this format, so that we easy can verify that it's a correct solution. The output should be using numbers for the symbols, separating them with a comma and a space. Here is an example of a locally-surprising sequence S=4, L=17: 0, 0, 2, 3, 3, 0, 3, 1, 2, 1, 1, 3, 2, 2, 0, 1, 0

5 The Harsh Reality of EAs

In general, the programming of EA solutions to hard problems involves two significant phases:

1. Coding the genome representation, genetic operators, genotype-phenotype mapping, fitness function, etc.
2. Tuning the parameters of the system to actually solve the problem

The second phase, tuning, often requires **much** more time than the first phase. Be aware of this when scheduling time for this (and other) EA assignments. Just getting the system to run, bug-free, is normally half (or less) of the total work.

6 Report

You should write a report answering the points below. The report can give a maximum score of 8 points. Your report must not exceed 4 pages in total. Overlength reports will result in a point being deducted from your final score. Print on both sides of the sheets, preferably. Bring a hard copy of your report to the demo session

a) Document your implementation. (2p)

- A clear, concise description of your EA code in text and a few diagrams.
- A justification of your code's modularity and reusability. You should describe how easy it is for your code to incorporate new phenotypes, genotypes, genetic operators and selection mechanisms as may be needed to handle new problems. Provide small examples of code that verify your claim.

b) An analysis of the performance on the One-Max problem. (2p)

- Run your EA on a 40-bit One-Max problem where the adult selection protocol is full-generational replacement and the parent selection mechanism is fitness-proportionate. First run the problem using various population sizes until you find the approximately minimal size that allows One-Max solutions to be consistently found in under 100 generations. Then, using only that population size, experiment with different values for the crossover and mutation rates. Use fitness plots to show the different results. Make a statement as to what the best choices are for these parameters in your runs.
- Using the best-found population-size, mutation and crossover settings from the previous experiments, do a new set of experiments to find the parent selection mechanism that gives the best results. Again, document your experiments with fitness plots.
- Modify the target bit string from all 1's to a random bit vector of length 40. Do you expect this to increase the difficulty of the problem? Run the EA and find out, using the best found configuration from above. Document your results from at least 4 different runs as part of your comparison.
- Run the EA on the 40-bit LOLZ problem with z=21, using the same configuration again. Run it at least 8 times, and document the results with fitness plots. Explain the results.

c) Surprising Sequences (2p)

- Draw a diagram of the genetic encoding used for this problem: genotypes, phenotypes, translation process. Include a very brief description.
- Describe (with text and a little mathematics) the fitness functions used to evaluate globally- and locally-surprising sequences.
- Create a table showing the best locally-surprising sequences that your EA finds for S = 3, 5, 10, 15 and 20. The table should include S, the population size used, the number of generations it took, L and the sequence found.
- Create a similar table for globally-surprising sequences, using the same symbol sizes (S).

d) Difficulty (2p)

- Given the for problems (One-Max, LOLZ, locally-suprising and globally-surprising), rank them in terms of their degree of difficulty (for an EA). Explain your decision.

The fitness plots for the tasks above should contain multiple runs. Because of the randomness, some runs will we worse than others. Don't cherry pick only runs with good results. We know how the graphs are supposed to look.

7 Demo

There will be a demo session where you will show us the running code and we will verify that it works. This session can give a maximum of 12 points.

For this demo, you will use your EA to find both locally and globally surprising sequences. For a given S and a few minutes of time, you should find as long sequence as possible.

It's a good idea to beforehand find ballpark figures of how long sequences (L) your algorithm can find for various S, so that you don't waste time during the demo looking for impossible L's or too easy L's.

8 Delivery

You should deliver your report + a zip file of your code on It's Learning. The deadline is given on the assignment on It's Learning. The 20 points total for this project is 20 of the 100 points available for this class. For this project you should work alone.