Homework Module:

Programming the Basics of an Evolutionary Algorithm(EA)

Purpose: Gain experience with the basic mechanisms underlying EAs by implementing and testing them.

1 Assignment

In the programming language of your choice (although C++ or Python is recommended), implement all of the basic components of an evolutionary algorithm and briefly test them on the One-Max problem. This homework module will normally (though not always) be combined with another module that involves a slightly more complex problem to solve with your EA, for example the Knight's Tour, the Short Taps puzzle or the Colonel Blotto game.

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.
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 generic EA, you only need to define them for binary genomes (i.e. bit vectors).
7. A basic **evolutionary loop** for running the GA through many generations of evolution.
8. A **plotting routine** that allows the user to visualize the results of an EA run. The key visual aid is a plot of the fitness progression, showing the maximum and average fitness (along with the standard

deviation) for each generation of the run. You may decide to dump the data to file and plot the fitness progression afterwards in another program, such as Python's matplotlib, Matlab, Excel or Gnuplot. There is no need to write the plot routine from scratch. Plots of fitness progression will be a standard deliverable for all homework modules involving EAs.

Aside from the plotting routine, all of the above aspects must be coded from scratch. If you are in doubt as to whether a given routine is *low-level* enough to be merely reused (instead of implemented by you), then consult the course instructor.

Your program should be highly parameterizable, so that factors such as population size, number of generations, crossover and mutation rates, etc. can be specified by the user at 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 program should be object-oriented, with basic classes for populations, genotypes, phenotypes, selection mechanisms, etc. 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.

1.1 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.

2 Deliverables

1. A clear, concise description of your EA code in text and a few diagrams **(2 points)**.
2. 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. **(2 points)**.
3. An analysis of the performance of 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. **(2 points)**

4. 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. **(2 points)**.

5. 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. Document your results from at least 4 different runs as part of your comparison. **(2 points)**.

Your report should be NO LONGER THAN 4 pages long, including fitness plots and other diagrams. **Write clearly and concisely. Reports longer than 4 pages may be penalized with point loss.**

Remember: the core of your program should be reusable on other evolutionary-computation homework modules!

2.1 Warning

Depending upon the actual course and semester - your instructor uses this module in various courses at various universities - you may or may not be required to do any or all of the following:

1. Demonstrate this module to the instructor or a teaching assistant.
2. Upload the report and/or code for this module to a particular site such as *It's Learning*.

Consult your course web pages for the requirements that apply.