Project 2: Genetic Algorithms

Complex Adaptive Systems
CS 423/523 Spring 2013
Professor Melanie Moses
UNM Computer Science

The Rules:

Turn in a hardcopy of your report at the **START of class on MARCH 6th.**

Turn in electronic copies of code and report as instructed on the course web page.

You will work in pairs. You should work with a different partner than you worked with on Project 1.

You may talk to other students and discuss the questions, but you may NOT share code with other teams to generate results or figures, and each team must come up with their own description of their work.

You may alter code you find online; however, you must DOCUMENT the source of all code that you incorporate into your analysis and report. If you extend, modify, or completely re-write code, anything from changing variable names to modifying the control structure of the code—no matter what you have done to the code, you MUST DOCUMENT the source of all code you did not write yourself. You must also explain what you did to modify any downloaded code in the Methods section.

You must cite the source (i.e. book, published article or web page) of any information you paraphrase in your report.

You may program in whatever language you like, but you must provide a README file providing exact steps to run your code. You must supply any libraries that are needed to make your code run on the CS machines.

I will not comment on draft versions of your reports by email. However, you are encouraged to come to office hours with questions, and you may bring a draft of your report.

PROJECT 2, PART 1

The first part of this project is a straightforward analysis of genetic algorithms. You will study the effect of varying types of crossover, mutation rates and elitism on two problems.

A basic genetic algorithm written in Matlab is available from [1].

By default this code implements 2-point crossover, with tournament selection, without elitism for the "maxones" problem. The maxones problem is a very simple fitness maximization problem in which the fitness score is the number of ones in a string, and the maximum fitness is the length of the string.

You may modify this code, or write your own code from scratch, or download and modify other code to implement your GA. Be warned however that there are many subtle differences in how GAs are implemented, so if you choose to modify other code, make sure you understand exactly how it is implemented.

First, you will test variants of the GA on the maxones problem. For all problems in Part 1, you should call the ga function with a population size of 80, a maximum of 100 generations, and chromosome length (string length) of 20.

You should also change the GA to implement roulette selection, with the probability of being selected for the next generation set to the square of the fitness score squared (this roulette function is included in the sample code).

Vary crossover in the following 3 ways: use 1-point crossover (with a crossover rate of 50%), 2-point crossover (with a crossover rate of 50%), and a crossover rate of 0 (no crossover). Note that the sample code has an implementation of 2-point crossover. You'll need to write your own 1-point crossover function.

Vary mutation rate as 0.001, 0.01 and 0.1.

The sample Matlab code does not use elitism. Repeat each experiment above with and without elitism. For the runs with elitism, the best individual of each generation should be copied unmodified into the next generation. Using elitism, the best fitness can never decrease over successive generations.

You should run a total of 18 experiments (3 kinds of crossover, 3 mutation rates, each with and without elitism.) Repeat each experiment 10 times. For each experiment, calculate a mean and standard deviation for the number of generations until optimal solution is found OR if the optimum is not found, mean best fitness reached in 100 generations.

Create a Table 1 that has three columns, one for each mutation rate (0.001, 0.01 and 0.1). The table should have 6 rows: the first 3 rows should show values for the 3 crossover types (none, 1-point, and 2-point) without elitism, and rows 4 – 6 should show values for the 3 crossover types with elitism. Report the results of each experiment as a mean +/- standard deviation for the number of generations to reach best fitness (G) or the fitness value (F) after 100 generations. Each cell in your table should contain either G = mean +/- stdev or F = mean +/- stdev. Include a 1 sentence caption explaining your table.

The second task for Part 1 of this project is to repeat the above analysis for a different fitness function, called Whitley's function [1,2].

$$f(x_1 \cdots x_n) = \sum_{i=1}^{n} \sum_{j=1}^{n} \left(\frac{(100(x_i^2 - x_j)^2 + (1 - x_j)^2)^2}{4000} - cos(100(x_i^2 - x_j)^2 + (1 - x_j)^2) + 1 \right)$$

$$-10.24 \leq x_i \leq 10.24$$

$$\text{minimum at } f(1, 1, \cdots, 1) = 0$$

This is a minimization function, meaning that the best fitness is the one that has minimum value. The string of all ones has the minimum value of 0.

You will need to change how you generate the population of individuals (each chromosome is 20 real numbers between -10.24 and 10.24), the fitness function (implement the above equation), the mutation function (you should describe in your methods how you mutate real numbers). Remember that this is a MINIMIZATION function—thus, your roulette selection algorithm should be biased to choose strings with lower fitness values for the next generation.

Store the results of your analysis in Table 2.

Your report should follow the same format specified for Project 1. It should include a brief introduction to genetic algorithms and all parts of your assignment. Methods should cite any downloaded code and specify your implementation decisions. Results should include the two tables, two figures (as described below) and answers to the following questions:

How much does elitism speed up the search for solutions to these 2 problems?

To answer to this question, you should create and refer to **2 figures,** one for the maxones problem, and one for Whitley's function. On each, graph 4 lines: average and best fitness versus generation, with and without elitism. Be sure to specify which crossover and mutation parameters were used to generate the data in your figure.

How much do 1-point and 2-point crossover speed up the search for solutions to these 2 problems?

What is the best combination of crossover and mutation rate for each problem?

Explain how the type of crossover, the mutation rate, and the use of elitism interact (i.e. should a different mutation rate be used for different types of crossover and use of elitism?)

Suggest a way to speed up your GA's search for fit strings. Run a set of experiments and explain whether (and how much) your improvement speeds up search. Explain any drawbacks to your approach.

Project 2, Part 2

Your job is to improve the performance of colonies in the GA-Integrated Ant (GIAnt2.0) model. This model simulates certain behaviors of seed harvesting ants, using a GA to fine-tune the behaviors to maximize seed collection rate [4]. The same simulation is used in [5] to control a swarm of robots.

The GIAnt2.0 model places two colonies on a field of seeds and evolves parameters to maximize the number of seeds collected in a specified number of time steps. By default the model has the colonies compete for seeds. The 'coop' flag can be set to evolve the 2 colonies to cooperate to maximize the total seeds collected together.

You have two specified tasks to complete, and one open-ended task. For tasks 2.2.1 and 2.2.2 change only the ga.cpp file. Problems 2.2.1 & 2.2.2 together will be worth 100 points, problem 2.2.3 will be worth 100 points (and Part 1 will be worth 100 points).

2.2.1 Plot fitness vs. generation time for competitive and cooperative colonies

Fitness is defined seeds collected either individually (while competing) or collectively (while cooperating). For this task do not alter the basic functioning of the GA. You'll need to store the number of seeds collected during each generation. **Show best and average fitness for the competitive and cooperative models.**

Also store the 'chromosome'—the set of best performing parameters for each GA for each generation. **Describe changes you see in the parameters over time. Do you see any interesting specialization in the two competing GAs?** For example, does one GA evolve colonies that use more pheromones than the other, or does one GA evolve a strategy to stay near the nest?

2.2.2 Alter the evolve function to increase performance

The code currently uses "independent assortment" of each parameter. That means that each parameter is randomly and independently chosen from one of the 2 parents. **Replace the current cross-over scheme with**

a) 1-point cross-over
b) another cross-over scheme of your choice, for example, 2 point crossover or n- point crossover or no crossover

Alter the way that real-valued parameters are mutated.

Alter the selection function (the code you were given currently uses tournament selection without elitism).

Explain: what you changed, <u>justify</u> why you made the particular changes you made, and describe your results using appropriate figures and text.

2.2.3 Change the code to evolve more fit colonies in *either* cooperative or competitive trials. Be creative. Describe your Methods clearly. Repeat your experiments with multiple runs and present data with statistics and figures to justify your conclusions. You are encouraged to research approaches that have been useful in prior work (just be sure to document your references). Your grade will be based on how well you justify your changes, how thoroughly you experiment to test the changes, how clearly you present results, and whether the change actually improves performance. If the changes don't improve performance, explain why.

You may change anything in the ga.cpp file. Additionally, for this task, you may change the way that colonies are chosen to compete or cooperate in main.cpp and placed on the field. For this assignment, you should not change the distribution of food or the definition of fitness. (You can do that for Project 3).

If you are enrolled in CS423, you may complete this part of the assignment by running experiments in which you vary a few simple parameters. If you are enrolled in CS523 you must do something that substantially alters the evolve function or changes the way that colonies interact in competitive or cooperative simulations.

We will spend time in each class until the due date answering questions, and discussing approaches for 2.2.3.

The code, a readme file, high-level descriptions of the code and rules for turning in your projects will be posted online. The code you turn in should run for a short number of interactions & generations, but your experiments will run longer. You must be **nice** when running jobs on the CS cluster. Your grade depends on careful analysis rather than long runs—be smart about how you allocate computational resources, and do not evolve a million colonies for a million generations!

REFERENCES

[1] http://cs.unm.edu/~melaniem/courses/CAS2010web/Site/Readings_files/ga.m (2/11/13).

[2] Jumonji , T et al "A novel distributed genetic algorithm implementation with variable number of islands", *Evolutionary Computation*, pp 4698-4705. 2007.

[3] http://www.cs.unm.edu/~neal.holts/GECCO2013/benchmarkFunction/whitley.html (2/11/13).

[4] Flanagan, T.P., K. Letendre, W.R. Burnside, G.M. Fricke and M.E. Moses. "How Ants Turn Information into Food." *Proceedings of the 2011 IEEE Conference on Artificial Life*:178-185. 2011.

[5] Hecker, J. P., K. Letendre, K. Stolleis, D. Washington and M. E. Moses. "Formica ex Machina: Ant Swarm Foraging From Physical to Virtual and Back Again." Proceedings of the Eighth International Conference on Swarm Intelligence, Brussels in *Lecture Notes in Computer Science*: 7461. 2012.