CEE 5290/CS 5722/ORIE 5340: Heuristic Methods for Optimization
Homework 3: Binary Genetic Algorithm

Assigned: Friday, September 12, 2014

Due: Friday, September 19, 2014 @ noon (free extension to Monday, Sep. 22 @noon)

1. If you wish to improve any of the basic approaches specified by the GA then feel free to do so – creativity is a small consideration in your final grade. In these cases, make sure you first answer the specific homework questions and then briefly describe, provide code and compare your new approach to the approach requested in the question. Provide this material in an appendix.
2. Marks may be deducted for a lack of neatness.

Reading: Section 3.1, 3.2 and 3.5.1 to 3.5.4 in the course package

The modified pseudocode from the text is as follows:

```
1  Procedure (Genetic Algorithm)
2        M = population size            % number of possible solutions at any instance
3        N_g = number of generations    % number of iterations
4        N_o = number of offspring      % to be generated by crossover
5        P_μ = mutation probability     % Also called mutation rate (M_r)
6        P ← Ξ (M)                      % Construct initial population, P
                                           Ξ is the population constructor

7  For j=1:M
8        Evaluate f(P[j])               % Evaluate fitness of all individuals
                                        % This is done before the GA algorithm is implemented
9  Endfor

10 Start Genetic Algorithm
11 For i=1:N_g
12       For j=1: N_o
13             (x,y) ← φ(P)             % Select two parents x and y from current population
14             Offspring[j] ← χ(x,y)    % Generate offspring by crossover of parents x and y (x=y is allowed)
15       EndFor
16       For i=1: N_0
17             Mutated[j] ← μ(y)        % With probability P_μ apply mutation all the offspring
18       EndFor

19 Evaluate fitness for offspring after crossover and mutation
20 P ← select (P, offspring)            %Select best M solutions from parents and offspring

21 EndFor

22 End (Genetic algorithm)
23 Return highest scoring configuration in P
24 End
```

1. This question will cover the optimization of a simple cost function using binary representation with Genetic Algorithms. Note that by convention, GAs are coded to maximize the fitness function.

Consider the 2-D cost function used in Homework 2:

F(s1,s2) = 10^9-(625-(s1-25)^2) *(1600-(s2-10)^2)*sin((s1)*pi/10)*sin((s2)*pi/10)

For question 1 here we will <u>**maximize**</u> this function. The global *maximum* is F(125,115) = 1088359375. Since each decision variable can take on integer values from 0 to 127, the input variables s1 and s2 can each be represented by 7-bit binary string. Hence the domain of this cost function can be represented by a 14-bit binary string as, $\underline{s}$ = [sb_1, sb_2, ... sb_{14}], where the first seven elements form the binary representation of s1 and the last seven bits represent s2. i.e.

$$s1 = \sum_{i=1}^{7} sb_i 2^{7-i} \quad \rightarrow \text{s1} = [sb_1\ sb_2\ sb_3\ sb_4\ sb_5\ sb_6\ sb_7] \rightarrow \text{e.g. } [0\ 0\ 1\ 0\ 1\ 1\ 1]$$

$$s2 = \sum_{i=1}^{7} sb_{7+i} 2^{7-i}$$

Note that the 14 bit string represents an 'individual' in the population (a solution).
The *Matlab* function "bin2dec" may be used to convert a binary string to decimal. Also dec2bin converts the other way.

a. Write a MATLAB function to implement a Genetic Algorithm GA.m. Please note that although there are a multitude of variations in coding a GA we ask that you follow the instructions given here as closely as possible so that results are generally consistent and grading is easier. The function header line should be:

```
function [solution, sbest]= GA(Xinitial, popsize, maxGen, pCrossover, pMutation)
```

where
solution will be a maxGen by 3 matrix where column 1 is generation number, column 2 is the average fitness of the population per generation and column 3 is the best fitness per generation.
sbest is the decision variable values of the best solution (not a binary string).
Xinitial is a binary matrix of size *popsize**14, with each row corresponding to a randomly generated initial point in the search space.
popsize is the constant size of the population in each generation.
maxGen is the total number of generations that the algorithm is to be executed for.
pCrossover is the value for probability of having crossover or not
If you want to provide other inputs or outputs in your GA.m coding (not required) this is OK but please describe/define any such variables in your code. You will need to write functions that implement the following as well:

fitness.m - to calculate the fitness of individual solutions (this is very simple code that calls your cost function and, if necessary, converts problem to minimization)
selection_R.**m -** which takes a population and generates parents from it using fitness proportional selection – Note that the selection procedure described in lecture is equivalent to using a weighted roulette wheel with slots proportional to the fitness of each member;
crossover.**m -** which performs single-point crossover at a randomly generated point along the string (14 bit here) and generates *two* children or offsprings per pair of parents; and

mutation.m - which mutates *each bit* (bit-wise mutation) in the string with a probability of *pMutation*.

Also code your GA.m function to use **Elitism**. Elitism is when the *m* highest fitness individuals (*m*<popsize) from the previous generation are always carried forward, undisturbed into the next generation. Here, use *m*=1 (only best individual is carried forward). Code this as you see fit in the GA.m file.
(Note:

- In pseudocode line 20 comments, this is just a general statement for GA. But for the purpose of the hw questions, please stick to the Elitism Rule mentioned above:
  - Generate No=20 children from 10 pairs of parents
  - Select the best parent to be carried forward into the next generation
  - Select the best 19 children and the best parent to be carried forward to the next generation
- You may try to put m also as an input in GA to make the code more general)

Even with the instructions above, there are various ways to implement the GA that vary according to how much of the coding is allocated to the GA.m file versus the functions called by GA.m. You are free to code the GA as you wish provided that you use the guidelines and functions above (fitness, selection, crossover, mutation) to help implement the required procedures. For example, crossover.m could input and output an entire population OR it could input a pair of parents and output a pair of offspring. Your code in GA.m that prepares the inputs for crossover.m would be different in these two cases.

b. Run GA 30 times on the fitness function described above with a population size of 20, for 50 generations, using a pCrossover=0.9 and pMutation=1/20=0.05 (You will experiment with GA parameters in a later homework). This is probably easiest if you *first* generate all 30 initial solutions sets and *then* call the GA.m within a loop 30 times, each time using one of the 30 initial solutions generated before the loop.

- Submit a plot showing the average fitness of the fittest member of the population at each generation (averaged over all 30 runs).
- Indicate if and how often you were able to obtain the global maximum in these 30 runs.

c. The fitness function in this problem is always non-negative.

- What problem might you encounter if the fitness function could take on negative values?
- How can this be corrected?

d. The implementation the crossover as described above could likely be improved for this particular problem.

- Give one idea you might consider and explain why you think it might be helpful for this problem.

(You do not need to implement computationally new crossover ideas at this time.)

e. The GA performed 1000 function evaluations per optimization trial in **b**.

- If one were to not use any heuristics, how many function evaluations need to be performed to be certain to find the optimal solution?

2. An alternative approach to roulette wheel selection is Tournament selection.

a) Code an alternative selection function called **selection_T.m.** This function will help implement binary Tournament Selection. Binary Tournament Selection is where two individuals are selected at random and compete (based on fitness) to become a parent and then the tournament is repeated to select the second parent. The above needs to be repeated until you have generated the required number of pairs of parents. You *may* need to modify some coding in GA.m for this depending on the way you coded GA.m from #1. You will probably find the 'randperm' function in Matlab helpful to do this.

- Submit your **selection_T.m** function and any part of the GA.m code *if* it has changed *via email* (you can call this file GA_T.m if it is new).

b) Under the same GA parameters as in #1 and the same initial populations as in #1, run the GA another 30 times using the selection_T.m function rather than the selection_R.m function. Use the same initial solutions as in Part 1(b)

- Submit a plot showing the average fitness of the fittest member of the population at each generation (averaged over all 30 runs) for both selection approaches (2 series on graph).
- Which one works better? And why?

3. In this question you will implement the GA to **<u>minimize</u>** the cost function in HW#2 and then compare it to the SA results you found with your best SA parameters in HW#2. The global minimum is F(125,125) = 891015625. If you did not save your SA results then you can simply redo your SA runs (this should be quick). If you haven't already, it may be a good time to figure out how to save algorithm output in Matlab (for example look at the Matlab 'save' function which saves matrices to text files).

a) Since your GA (and GAs in general) maximize the fitness function, you will need to convert the maximization problem to a minimization problem by modifying the fitness.m function accordingly. You can use any simple method you see fit to do this conversion. Submit your modified fitness.m function *via email* and describe in words your conversion.

b) Using the same GA parameters used in Question 1 run the GA 30 times starting from 30 new random initial populations. Compare the GA results to the SA results you found in HW#2 by submitting a plot of the average best cost function value (averaged over 30 trials in both cases) versus the number of *cost function evaluations* for each algorithm (both on same plot!). To do this, you will need to do 2 things:

- Modify the fitness function such that it now finds the minimum of the cost function (part(a))
- Change the generation numbers in your GA *output* to the corresponding number of cost function evaluations by multiplying them by the population size (20).

c) In a table, for both algorithms, report after having calculated 1000 cost function evaluation the average best cost function value, the standard deviation of the best cost function value and the number of times each algorithm finds the minimum for this problem.

d) Give one reason (there are more than one) why it is difficult to conclude based only on the results requested above that, in general, one algorithm is better than the other for this particular problem.

<u>Please remember to submit all requested scripts to Blackboard. Everything including the m-files (graphs, written responses to questions, etc.) must be submitted in hard copy into the homework box located in 220 Hollister.</u>