Report on Evolutionary Intelligence

(Genetic Algorithm)

Hamza Liaqat

20100182

Instructor: Dr. Yasir Mehmood

Lahore University of Management Sciences

The evolutionary Intelligence I used in this Assignement is what we covered in class: Genetic Algorithms.

The Robot is allowed 28 cycles only and to perfectly traverse the wall it needs at least 28 cycles. Therefore, it should perform each step correctly. For example, even if it makes one mistake, it will waste at least 1 step; Thus, never reaching the destination in prescribed limit.

To encode path, I used a string of bits. Since, a robot is allowed to perfrom 4 actions at a time. Four bits are needed to completely encode 4 actions. Any arbitrary encoding will work well; however, the encoding I used is the follwoing:

* 00 -> *Do Nothing*
* 01 -> *Step forward*
* 10 -> *Turn Right*
* 11 -> *Turn Left*

Note that to perfectly traverse the wall, the robot needs to take 28 correct steps. At each step, it has four options (2 bits for each step). Therefore, To perfectly encode the correct path from start to end, fifty eight total bits are required (58=28 \* 2). According to the above encoding, the correct path encoding is the following Holy BitString:   
  
'*010101010110 01101101010111011001100101010101100101010101'*.   
  
Green pairs represent marching forward, blue turning right, and red turning left. Pair for 'Doing nothing' was not used in the bit String encoding.

The goal is to learn this Bit String through *'evolutionary intelligence'* using an initial population.

1. The initial population consists of 56bit random Bit Strings.
2. To implement in python, I used BitString Library. Note that if you've not Bitstring library downloaded in your computer use either of the following pip/conda commands to download it:
   * pip install bitstring
   * conda install -c bioconda bitstring
3. For Finding the fitness of a candidate solution, I used a very strict condition: *if a robot, makes 1 wrong move, it will not be allowed to advance any further, and its fitness so far will be returned*. For example, on the contrary, a lenient condition would have been to allow the robot advance even if it makes wrong moves.   
   * The downside of using the strict condition is that a large population size is needed to find the optimum path in reasonably short interval of time. One may have to wait more than 15 minutes if small population size is used.
   * The bright side of this strict condition is its simplicity. No excessive states of grid or current direction of the robot at every instant need be maintained.
4. The core function in script is genetic\_alg() which is a replica in Python of pseudo code given by Larry Bull. The rest are helper functions; When necessary, their functionality is well docummented/commented. For mutating, recombining, guidlines of Larry Bull were followed exactly. For example, Recombine at (60-90)% of Chromosome Length (BitString Length). Mutate each bit with probability between 1/Population Size and 1/Chromosome length.
5. When you run Python script, you’ll be prompted to tell whether you would like to see the fitness of each candidate solution as candidate solutions evolve. Note that since Input/Output is expensive, it might take slightly longer to reach the optimum solution when outputing fitness of each candidate.
6. Also, note that for your convenience I have provided an additional file Jupyer Notebook which is identical to Python script but it will run in your Browser. I developed this whole assignment in Jupyter Notebook, and later converted it to separate Python script and this Report.