CS 790K: Genetic Algorithms
Homework Assignment # 0

Beifang Yi*

November 12, 2002

*Email: b_yi@cs.unr.edu

1. **The Problem**
Given a function *double eval(int *vec)* in which the input *vec* is an integer array with 100 as its size and 1 or 0 as its element, maximize the function's output by feeding *vec* with a sequence of 1's and 0's. Find the largest value and the corresonding *vec[0::99]* (the sequence of 1's and 0's).

2. **Problem Solving**

   - Problem exploration
   There are 2^{100} combinations of 1's and 0's for the input *vec* sequences. It has been estimated that 2^{100} is more than the total number of atoms in this whole universe and that even the fastest computer and quickest traditional algorithm can not go through all those sequences **forever** (because of our universe's limited life expectancy: less than one hundred billion years). Traditional algorithm can not help much for this black-box problem and at this time GA is not allowed. I have to use explore-and-exploit strategy for this problem.
   - Strategy
   As said above, no traditional method can be used for this problem in a short time. Only wild guess and good luck can help. But it would be always better, before our going into the wilderness, that we test the function with some certain designed sequences to see how much useful information we can get from the results. I enumerate the this explore-exploit progresses as follows:
     - **Define the input *vec* with size of 200 instead of 100:**
     The last 100 elements are the copy of the first part. When calling *eval(vec)* we transfer the parameter in the form of &*vec*[*i*] instead of *vec*. Thus for most cases when we produce a sequence we can feed the function with &*vec*[*0*], &*vec*[*1*]...&*vec*[*99*], saving much more time than producing another 99 sequences.
     The results proved this method most rewarding: by shifting a sequence of 1's and 0's, we can obtain useful information about 1's (or 0's) distribution on the *vec*.
     - **Produce the test (input) sequences of 1's and 0's:**
     It will be time-consuming, difficult, (and unreasonable) to design an algorithm of producing a set of *reasonable* input sequences of 1's and 0's. Randomly producing 1's and 0's may be more convenient and suitable in this case. But it will be a waste of time for one sequence

to call 100 random functions. My thought was: 1). given the number (*numOnes*) of 1's in the sequence; 2). randomly produce the *locations* of those 1's in *vec*; and 3). shift the sequence 99 times, resulting in 100 sequences. It is possible that there exist same sequences in the 100 sequences, but even under such circumstances it will be time saving compared with the time spent on the algorithm and extra random function calls.

- Explore-and-Exploit
  - **In case of maximum=100:**
    First, we get the function value of 50 for the sequences of all 0's and 1's.
    Then, randomly distribute small number of 1's on the *vec* and get 50 ± 10. Thus I assumed 50 as the base value.
    Therefore, I used one, two, three 1's on *vec* and examined the results: with 1's appearing on the left side more often there comes a higher returned value.
    Now, I tried to use the magic number 0.618: assign the rightmost 38 elements of *vec* as zero's and 45 ($\approx 68 * 0.618$) one's randomly in the left 68 vacancies. The returned values became larger, but not satisfying.
    As a last straw, I crossed out the random function and it is by chance that my sequence was 45 one's and 55 zero's. My program shifted the sequence and voila: I got 95.
    Finally I tried 50 one's and I got the maximum. It is a crazy sequence of which I had never thought!
  - **In case of maximum=101:**
    From the experience in the above case, I quickly determined that the first 10 elements of *vec* did **not** contribute to the returned value. I got the value of 90 with rightmost 50 zero's and 40 one's in between.
    In two cases: with 10 one's and 10 zero's for the first 10 elements, I randomly distributed 1's, the number of which was from 13 to 23, to the right 50 vacancies. Computer ran for two days and nights with nothing improved.
    Then I let the first left 10 and rightmose 50 elements randomly take 1's. These two parts **independently** chose their 1's. And voila again: the returned value larger than 90! This indepence is so important: there are **only** 1024 different sequence for the first 10 elements

and computer can go through them all very, very quickly!
I examined the sequence: with 1010101010 as the leftmost elements.
So with the rightmost becoming 0', I got the maximum—hooray!

3. **Conclusion**
   Mostly it was lucky that I found the two sequences. But the strategy of shifting a certain number of 1's (either in random place or not) did help much in the exploring process. On the other hand, because I did not design a proper searching algorith, I could have time for analysis of the returned results from a specific sequence and then quickly modified my program for another sequence pattern.

4. **The Code**
   Attached are a C++ class and several test drives. Those are very short and simple.