BioComputation

Worksheet 2: Simple Genetic Algorithm Search Operators

In a computer language of your choice, you're implementing a simple genetic algorithm (GA). That is, a generational GA which uses a binary encoding, tournament selection, single-point crossover and bit-wise mutation (refer to lecture 2). Worksheet 1 covered the basics of implementing a population and running selection. Remember the full algorithm:

```
BEGIN
  INITIALISE population with random candidate solutions;
  EVALUATE each candidate;
  REPEAT UNTIL ( TERMINATION CONDITION is satisfied ) DO
    1 SELECT parents;
    2 RECOMBINE pairs of parents;
    3 MUTATE the resulting offspring;
    4 EVALUATE new candidates;
    5 SELECT individuals for the next generation;
  OD
END
```

After selection, the temporary offspring population is mostly full of copies of the better solutions – but no new solutions. Crossover and mutation are needed.

For every pair of individuals in the offspring population, pick a random point in their length and swap over the genes in the "tails" to form two new solutions for one-point crossover:

```
individual temp;
for( i=0; i<P; i+=2) {
        temp = offspring[i];
        crosspoint = random()%N;
        for( j=crosspoint; j<N; j++ ) {
                offspring[i].gene[j] = offspring[i+1].gene[j];
                offspring[i+1].gene[j] = temp.gene[j];
        }
}
```

Mutation needs a per-bit probability to be defined and then test each gene of each offspring:

```
for( i=0; i<P; i++) {
        for( j=0; j<N; j++ ) {
                if( random() < MUTRATE ) {
                        if( offspring[i].gene[j] == 1 ) offspring[i].gene[j] = 0;
                        else offspring[i].gene[j] = 1;
                }
        }
}
```

Or mutation in Python:

```
for i in range(  0, P ):
        newind = individual();
        newind.gene = []
        for j in range( 0, N ):
                gene = offspring[i].gene[j]
                mutprob = random.randint( 0, 100 )
                if mutprob < (100*MUTRATE ):
                        if( gene == 1):
                                gene = 0
                        else:
                                gene = 1
                newind.gene.append(gene)
```

Once a new population of solutions has been made, the fitness of each should be calculated, as before. Then the old population is overwritten with the new population (use a “deep” copy).

Useful information to record per cycle/generation of the GA is the best fitness in the current population and the mean fitness in the population. The former should go up in a step-wise fashion whereas the latter should continually rise towards to the former.

Once you are happy this is working, attempt to maximise the best fitness achieved by your GA over fifty generations for a fifty-bit genome in a population of fifty individuals. That is, explore the effects of altering the probability of crossover and mutation events occurring. Refer to the lecture notes for some guidance on default settings. Population and tournament size can be varied too. Remember this is a stochastic algorithm so results should be averaged over (preferably) ten separate runs (different random seeds) for a true estimate of utility.