CSC-20043 Computational and Artificial Intelligence I
Evolutionary Algorithms Practical 1

The following is not a typical GA. It is used here to provide a very simple first example of a GA. The fitness function is interesting but you can complete this practical without understanding it or reading the paper below.

Hinton and Nowlan's Experiment

Genome length: 20
Alleles: 0,1,?
Fitness function (you do not need to worry about this):
  Learning: Set each ? randomly to 0 or 1
  1000 learning trials per evaluation
  Only one correct answer: all bits are 1s
  Needle in a haystack
  Very hard for evolution alone
  Fitness = (1+19n)/1000
  n = number of trials remaining after target pattern found
Generational GA using roulette wheel selection
Population size: 1000
Result: Solution found rapidly

Code outline of GA forHinton and Nowlan's Experiment

The code outline on the next page is available as file adcHintonNowlan-outline.java

The missing parts are for you to complete (this is the work for Evolutionary Algorithms Practical 1) to check your understanding. (This is not assessed, and you do not need to submit anything.)

Instructions:
A) Read through the code outline and understand what each part should do
B) Write code to replace the BBBBs, to complete the code that selects parents for reproduction
C) Write code to replace the CCCCs, to produce the child genotype using crossover
D) Run the completed program several times, and compare your output to figure 2 from the paper.
  https://content.wolfram.com/sites/13/2018/02/01-3-6.pdf
  (You might like to graph your results, e.g. using spreadsheet software.)

More about Hinton and Nowlan's Experiment

The real importance of Hinton and Nowlan's paper "How Learning Can Guide Evolution" is that it demonstrates how learning and evolution can interact:

- The Baldwin Effect
  - No inheritance of acquired characteristics (Lamarck was wrong)
  - But lifetime learning can guide evolution
- Hinton and Nowlan were the first to demonstrate this in a computer model
  - “We demonstrate that this effect allows learning organisms to evolve *much* faster than their non-learning equivalents, even though the characteristics acquired by the phenotype are not communicated to the genotype." (from their abstract)
- You can read the paper here: https://content.wolfram.com/sites/13/2018/02/01-3-6.pdf

```java
import java.util.Random;
class adcHintonNowlan {

final int numGenotypes      = 1000;
final int genotypeLength    = 20;
final int numLearningTrials = 1000;
final int numGenerations    = 50;

Random rng = new Random();
int random(int n) { return rng.nextInt(n); } // random integer between 0 and n-1

final int zero=0, one=1, questionMark=2;

int genotypes[][] = new int[numGenotypes][genotypeLength];
int fitness[] = new int[numGenotypes];
// fitness = 1000*"1+19n/1000" where n = number of learning trials remaining
//             or rather numLearningTrials + 19 * n

void calculateFitnessArray() // calculate fitness[] *YOU DO NOT NEED TO WORRY ABOUT THIS*
{ for (int i=0;i<numGenotypes;i++)
  { int numTrialsRemaining;
    for (numTrialsRemaining=numLearningTrials;numTrialsRemaining>0;numTrialsRemaining--)
    { int g;
      for (g=0;g<genotypeLength;g++)
        if (genotypes[i][g]==zero || (genotypes[i][g]==questionMark&&random(2)==0))
          break;
      if (g==genotypeLength) break;
    }
    fitness[i] = numLearningTrials + 19 * numTrialsRemaining;
  }
}

void nextGeneration()
{ calculateFitnessArray();

  int sumFitness=0; for (int i=0;i<numGenotypes;i++) sumFitness += fitness[i];
  int newGenotypes[][] = new int[numGenotypes][genotypeLength];
  for (int i=0;i<numGenotypes;i++) // for each new genotype
  { int parentA, parentB;

    // Select parents for reproduction.
    // The probability of a parent being selected should be proportional to its fitness.
    // Hint: calculating random(sumFitness) will help

    BBBB

    int cutPoint = 1+random(genotypeLength-1);

    // Crossover: newGenotypes[i] should copy genes before the cut point from
    // genotypes[parentA] and the from the cut point on from genotypes[parentB]

    CCCC
  }
  for (int i=0;i<numGenotypes;i++)
    for (int g=0;g<genotypeLength;g++) genotypes[i][g] = newGenotypes[i][g];
}

void outputStatistics(int generation)
{ int counters[] = new int[3];
  counters[0]=counters[1]=counters[2]=0;

  for (int i=0;i<numGenotypes;i++) for (int g=0;g<genotypeLength;g++) counters[(int)genotypes[i][g]]++;

  double d=numGenotypes*genotypeLength;
  System.out.println(""+generation+" "+counters[0]/d+" "+counters[1]/d+" "+counters[2]/d);
}

void run()
{ for (int i=0;i<numGenotypes;i++) for (int g=0;g<genotypeLength;g++) switch (random(4))
  { case 0 : genotypes[i][g] = zero         ; break; //25%
    case 1 : genotypes[i][g] = one          ; break; //25%
    default: genotypes[i][g] = questionMark ; break; //50%
  }
  outputStatistics(0);
  for (int generation=1;generation<=numGenerations;generation++)
  { nextGeneration();
    outputStatistics(generation);
  }
}

public static void main(String args[])
{ new adcHintonNowlan().run();
}

}
```