Soft Computing Laboratory

Task number 3

Simple Genetic Algorithm

Florian Tissier, 901415
Geoffrey Ramdé, 901414

1 The main goal

We wish to implement a Simple Genetic Algorithm (SGA) in order to optimize the function $f(x) = (e^x \sin(10\pi x) + 1)/x + 5$ on the $[0.5, 2.5] \subset R$ interval in which the function takes positive values.

2 Implementation

The implementation is made in C++. The parameters that are used in it are:

- Number of generations N = 60
- Number of chromosomes M = 40
- Crossing probability PC = 0.4
- Mutation probability PM = 0.2
- Reproduction probability PR (implicit) = 1 - PM - PC = 0.4

2.1 Roulette Wheel Selection

```cpp
Chromosome* Population::choose()
{
    calcF();
    for(int j=0;j<M;j++)
        calcProbas(j);
    for(int j=0;j<M;j++)
        calcDistris(j);
    int i, index=0;
    double r=(double)rand()/(double)RAND_MAX; //random between 0 and 1
    for(i=1;i<M;i++)
    {
        if(chromosomes[i]->distribution>r && chromosomes[i-1]->distribution<r)
        {
            index=i;
            break;
        }
    }
    return chromosomes[index];
}
```

2.2 Fitness Function

```
void FitnessFun::calculate(Chromosome* input)
{
    input->fitness=((exp(input->value)*sin(10*(input->value)*M_PI)
    +1)/((input->value)+5));
}
```

2.3 Crossing

```
if(r<=PC)
                {
                    //perform crossing
                    int j=rand()%NBBITS;
                    int tempBits[NBBITS-j];
                    Chromosome* temp1=current->choose();
                    Chromosome* temp2=current->choose();
                    for(int k=0;k<NBBITS-j;k++)
                        tempBits[k]=temp1->bits[j+k];
                    for(int k=0;k<NBBITS-j;k++)
                        temp1->bits[j+k]=temp2->bits[j+k];
                    for(int k=0;k<NBBITS-j;k++)
                        temp2->bits[j+k]=tempBits[k];
                    temp1->calculateValue(temp1->bits);
                    temp2->calculateValue(temp2->bits);
                    next->chromosomes.push_back(temp1);
                    next->chromosomes.push_back(temp2);
                    i+=2;
                }
```

2.4 Mutation

```
else if(r>PC && r<=PC+PM)
                {
                    //perform mutation
                    int j=rand()%NBBITS;
                    Chromosome* temp=current->choose();
                    temp->bits[j]=1-(temp->bits[j]); //flip the
    bit
                    temp->calculateValue(temp->bits);
                    next->chromosomes.push_back(temp);
                    i++;
                }
```

2.5 Reproduction

```
else{
                    //perform reproduction
                    Chromosome* temp=current->choose();
                    next->chromosomes.push_back(temp);
                    i++;
                }
```

2.6 Main

Two populations are created initially ("current" and "next" that is empty), we then have two loops "run" and "gen".

In each generation we calculate the fitness function of each chromosome before choosing a random number to decide the probability of crossing, mutation or reproduction. In any case the Roulette Wheel Selection choose the best chromosome. We however have to check the number of chromosomes as the reproduction process add another one. If it's the case, we delete it.

The process is done M times to generate all the chromosomes in the "next" population.

The "next" population is transferred in the "current" one and the gen loop is reiterated. The best solution of each "gen" loop is saved in a table. Once the gen loop is finished, we display the best solution of the "run" loop.

3 Results

To verify our results, the program is run 5 times and we keep the result of the last run loop.

First Time	2.45179
Second Time	2.44798
Third Time	2.49953
Fourth Time	2.43135
Fifth Time	2.44162

The results are close to 2.45, the researched value.