TRINITY COLLEGE DUBLIN, SCHOOL OF MATHEMATICS

# Assignment #2, Module: MA5611

Gustavo Ramirez

January 24, 2017

## 1 Problem description

The goal of this assignment is to develop a framework for implementing genetic algorithms in parallel using MPI. Then using this framework you will develop strategies for playing the iterated prisoner's dilema.

## 1.1 Solution

- **Task 1**

  For the implementation of the genetic algorithm:

  1. generate the population with its values randomly set. This is only done once
  2. calculate the fitness for all the members of the population combined and separated
  3. calculate the probabilities of being selected, each of them proportional to their fitness, and normalized. These probabilities are useful in the next step, in the sense that when a chromosome is to be selected, it is done through the wheel roulette selection.
  4. crossover step. X% of crossover rate means that percentage of the next generation is composed of crossover offspring (and for creating two new elements of the offspring, two elements of the current generation are chosen through wheel roulette selection and then crossed), and the rest of the empty places are filled in doing wheel roulette selection again. The two chromosomes from the current generation crossover after first bit to give the new offspring. **If the population is $n$, and the fractional crossover rate is $c_r$, then this step loops $nc_r$ times; in each iteration of this loop, a pair of current chromosomes is selected, crossed, and the offspring created and stored. Additionally, it loops $n - nc_r$ more times, to fill the empty places.**
  5. mutation step (acts over the just created offspring). Y% of mutation rate means to flip each bit with a probability Y%. **If the population is $n$, the chromosome size $s$, and the mutation rate $m_r$, then this step loops $nsm_r$ times, but without flipping two times the same bit. For this flipping process, all of the $ns$ bits of the population are taken into consideration.**

  Perform steps 3, 4 and 5 $n_g$ times, where $n_g$ is the number of generations.

- **Task 2**

  In the previous task, the framework for the implementation of genetic algorithms for certain fitness function, was done.

  In this second task, the PD was implemented as fitness function. For this, each chromosome was assigned 16 bits; each bit represents one of the 16 possible outcomes of two previous games of the PD. Initially, the GA framework sets the values of all the chromosomes randomly, which means that the whole population has random rules to play the PD based in the previous two games.

Another important detail is that, in the implementation of the fitness function, each chromosome plays a PD game with each of the other chromosomes in the population, from which a good representative fitness is obtained (depending on the kinda of "person" that each chromosome represents, and thinking in terms of years in jail skipped, that will determine how good each chromosome is, i.e. how prone to be out of jail is).

- **Task 3**

  To parallelize the GA framework, the usual and simple scheme of Master-Slave was used.

  Specifically, the implementation using PD as fitness function was the parallelized code, as is a more interesting example than the sum of 1s in a string.

  The most relevant steps in this parallelization are:

  1. the master initializes the chromosomes data
  2. the master sends the chromosomes' data to each slave process
  3. the master does its corresponding work
  4. the slaves do their corresponding work
  5. the slaves send fitness values back to the master process
  6. the master gathers those fitnesses, and then performs crossover, random selection and mutation
  7. re-start again from point 2, during all the generations

  Comparison between a serial and a parallel execution in this case:

  Specs of system:

  number of plays of PD: 12, population size: 500, mutation rate (percentage): 5, crossover rate: 80, number of generations: 100

  **SERIAL CODE** with **execution time**: 0m52.468s.

  **PARALLEL CODE** with **execution time**: 0m10.767s