COMP6026 - Assignment 2 - Group Selection

Henry Lovett - hl13g10

January 5, 2014

1 Introduction

In an evolutionary sense, being selfish is advantageous as an individual can reap the benefits of being in a group without any contribution. A basic example of this is the Prisoner's Dilemma . However, in nature, cooperation is common. This then raises the question of why cooperation exists.

In Powers et al. (2007), a situation was set up with selfish and cooperative individuals. Each individual also has a preference of being in a small or large group. Resources were allocated to the groups and the population increased depending on the amount of resources the genotype had. Selfish individuals had a higher growth, but higher consumption of the resource than the cooperative. A small group had less resources per captia than the large group.

In each generation, the pool was split into as many small and large groups as possible and allocated resources. The numbers of the genotypes were then allowed to grow.

By doing this, it was shown that the genotype of small cooperators flourished and became the only genotype in the population.

This paper discusses the reimplementation of the experiment Powers et al. (2007) and a comparison of results in sections 2 and 3. An extension to this work is covered in section 4, the results of which are shown in section 5 and section 6 concludes the paper.

2 Reimplementation

- Initialise
- for number of generations:
  - Make groups
  - for timesteps:
    - Allocate resources
    - Grow populatiosn
  - Reform migrant pool
  - Scale migrant pool

Parameter, *symbol*	value
Cooperative consumption rate, C_c	0.1
Selfish consumption rate, C_s	0.2
Cooperative growth rate, G_c	0.018
Selfish growth rate, G_s	0.02
Population size, N	4000
Small group size, N_small	4
Large group size, N_large	40
Number of generations, N	120
Number of timesteps, t	4
Resource for small groups, R_{small}	4
Resource for large groups, R_{large}	50
Death rate, K	0.1

Table 1: Parameters used in the reimplementation

This experiment used individuals with two genotypes, giving four possible combinations of individual. The genotypes were whether the individual preferred small or large groups, and whether it was selfish or cooperative. The four possible combination therefore were: cooperative & small, selfish & small, cooperative & large, selfish & large.

In the reimplementation, an individual was not explicitly represented. Instead, a list of four values was used to store the total number of each genotype. This was used for both the migrant pool and the groups.

Initialisation was done exactly proportionately. Each genotype was assigned $N/4$ number of individuals.

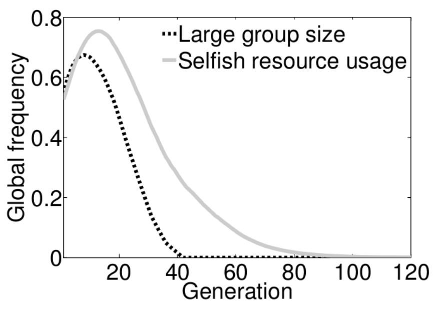
During each generation, the migrant pool was split down into as many small groups as possible. Individuals were split into either small or large groups depending on their preference in their genotype. Groups were made to represent the proportions of the global migrant pool. Only full groups were allowed and any members left over from group allocation were removed from the population. These groups were then allocated resources and allowed to grow.

Resources were allocated to each genotype proportionately,The assumed knowledge is difficult to gauge given the amount depending on their genotype. This was done using equation (1). It is biased to allocate more resource to the selfish genotype ($0.02 \times 0.2 > 0.018 \times 0.1$). R also changes depending on the group size - small groups have limited resources to encourage cooperation, and large groups have more resource per captia.

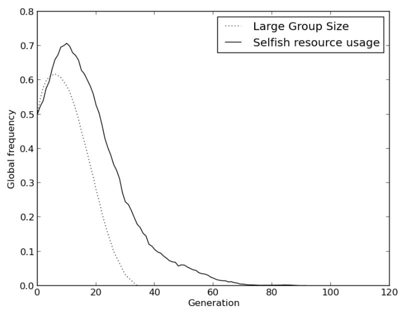
$$r_i = \frac{n_i.G_i.C_i}{\sum_j (n_j.G_j.C_j)}.R \tag{1}$$

Once the resources are allocated, the groups are then grown. The new population size is calculated by three terms, seen in equation 2. The first is the current size and the third is a constant death rate to all genotypes. The middle term uses the resources allocated and the consumption rate of the genotype. As the consumption for a cooperative genotype is lower, this is biased to grow the cooperators more.

$$n_i(t+1) = n_i(t) + \frac{r_i}{C_i} - K.n_i(t) \tag{2}$$

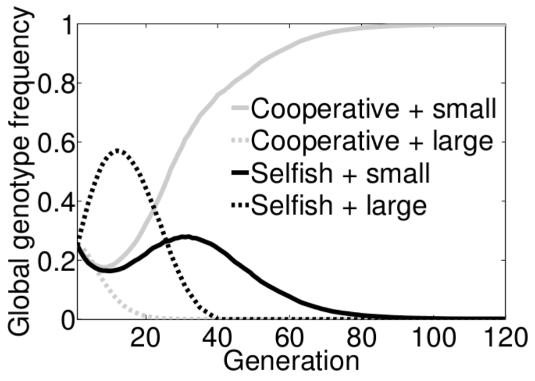


(a) The original results from Powers et al. (2007).

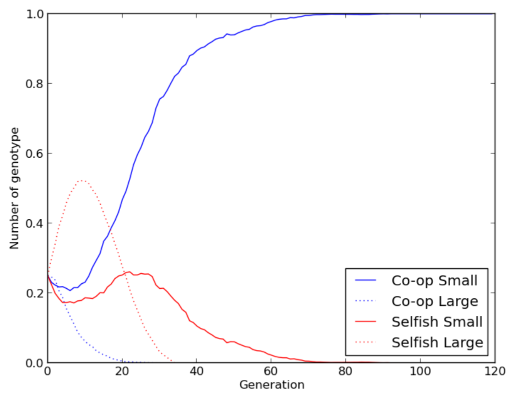


(b) Reproduced results.

FIGURE 1: Average environment and strategy through time.



(a) The original results from Powers et al. (2007).



(b) Reproduced results.

FIGURE 2: Change in genotype frequencies over time.

3 Comparison of results

The reproduced results were found to be very close to the original data. Figure 2 shows the proportions of each genotype in the population. In both graphs, the cooperators in the large group get immediately out competed by the selfish, and are then pushed to extinction. The numbers of large selfish then begin to diminish and both small genotypes increase before the cooperative small genotype excels and results in being the entire of the population. The population reached a steady state by 100 generations.

Figure 1 shows the proportions of the strategies. Both results show that the large populations reach 0 first and the selfish gene takes a little longer to be removed from the population.

The results obtained from the extension proved to be a very close replication to the original data, and therefore can be used for an extension of this work.

TABLE 2: The extra parameters used to implement the extension

Parameter, *symbol*	value
Proportion of population in middle group, $M_{proportion}$	$0.0 - 0.24$
Medium group size, N_{med}	22
Medium group resource, R_{med}	27

4 Extension

Discrete groups do not always occur in nature. The extension covered here adds a third middle group to the experiment. The middle group contains all genotypes in the pool in the same proportions. As before, an individual may only exist in one group. If it is in a middle group, then it cannot be in a small or large group.

The main parameters remain unchanged (apart from the number of generations, which was increased to 200) from Powers et al. (2007). This experiment set out to find when, if at all, the small cooperators were out competed by another genotype. It is predicted that the large selfish genotype will take advantage of this middle group once a large enough proportion of the population is placed in this.

NEED TO CITE SOME STUFF.

Some extra parameters were added to characterise the middle group. The size of this group, and the resources allocated, was made to be the average of the small and large group's size and resources. This was done to keep the same amount of resource per captia in the group. The final parameter was the parameter under test - the proportion of the population that was placed in the medium sized group. The parameters are summarised in table 2.

5 Results

A sweep of the middle proportions was done from 0 to 0.24. At each value of $M_{proportion}$, the simulation was run 10 times. After each simulation, the genotype with the largest population was deemed to be the 'winner' and a tally was kept. The number of wins of each genotype was plotted against the value of $M_{proportion}$ and can be seen in figure 3.

The results show that the small cooperators win consistently until around 0.03. From this point, the selfish large genotype starts to win some of the simulations. By 0.06, large selfish starts to win the majority of the simulations.

Both small groups win the occasional game in the higher proportions. This is assumed to be noise as no explicit checking was done to verify the populations had reached fixation.

The graphs in figure 4 show some of the populations in some of the simulations. The two graphs, 4(b) and 4(c) show two different outcomes for the same $M_{proportion}$ value.

6 Conclusion

This paper covered the reimplementation of the work in Powers et al. (2007). The implemented code was then shown to match the results expected and this then provided a platform to extend this work.

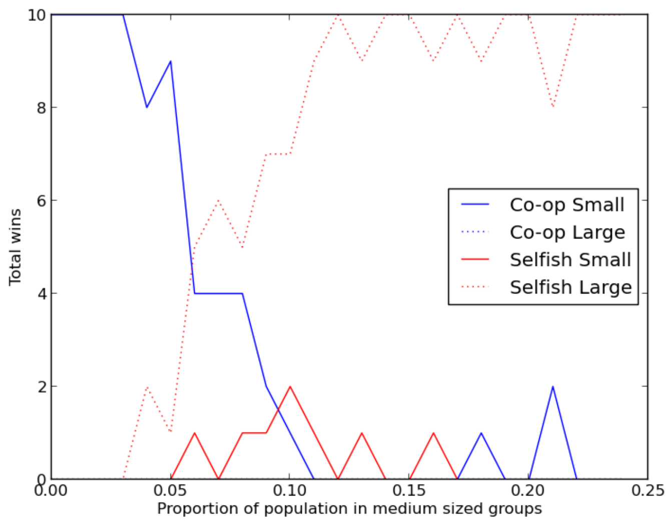
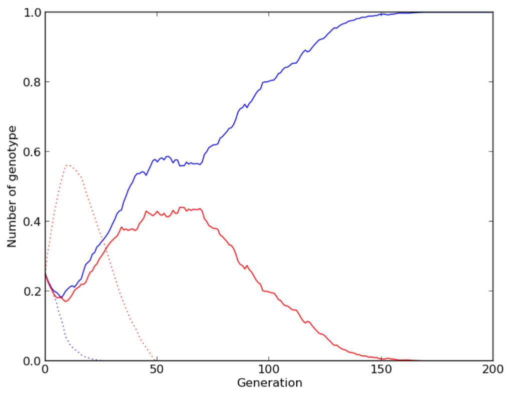
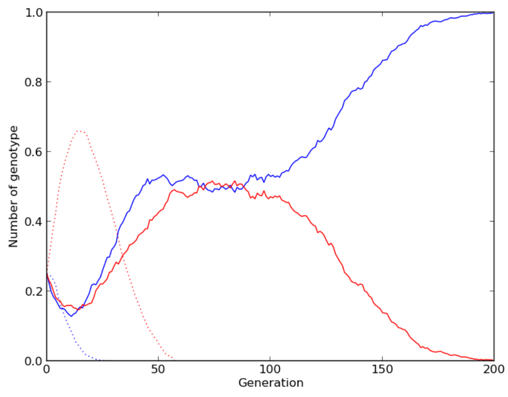


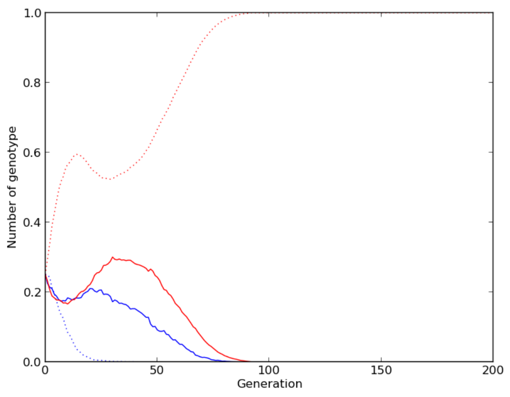
FIGURE 3: A sweep of the proportion of the population allocated into medium sized groups.



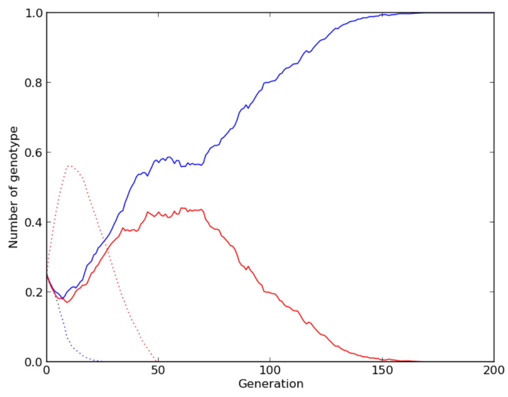
(a) Middle Proportion = 0.05



(b) Middle Proportion = 0.06 where the small cooperative wins.



(c) Middle Proportion = 0.06 where the large selfish wins.



(d) Middle Proportion = 0.24. The large selfish genotype wins outright.

FIGURE 4: Proportion of genotypes in simulations with varying proportions in the middle group.

The extension investigated set out look at creating more groups as a step towards making a fully continuous simulation. This was done by adding a middle group, for all genotypes being able to be a part of. A proportion of the pool was then allocated into these middle sized groups before allocating to the small and large sized groups.

It was predicted that the selfish gene would eventually take over the population as it would make use of being a defector in the middle sized group. This was found to be correct, as at about 0.06% of the population in the middle group, the selfish genotype won the majority of the simulations.

However, this work still creates discrete groups. Future work could improve this to have more groups, or even implement groups of many different sizes.

References

Powers, S. T., Penn, A. S., and Watson, R. A. (2007). Individual selection for cooperative group formation. In *Advances in Artificial Life*, pages 585–594. Springer.

7 Code

7.1 main.py

```
#!/usr/bin/python
# COMP6026
# @author hl13g10
# @brief reimplementation of Powers (2007) work for COMP6026 coursework
import math
import random
import pylab
outfile = "pool.txt"
fig = pylab.figure()
## Globals
K = 0.1 ## Global Death rate
R_small = 4.0 ## Resources for a small group
R_large = 50.0 ## Resources for a large group
Gc = 0.018 ## Growth rate for a cooperator
Gs = 0.02 ## Growth rate for selfish
Cc = 0.1 ## Consumption rate for a cooperator
Cs = 0.2 ## Consumption rate for selfsih
N = 4000 ## Population size
N_large = 40 ## Number of individuals in a large group
N_small = 4 ## Number of individuals in a small group
T = 120 ## Number of generations
t = 4 ## Number of time steps in groups

#The numbers will be stored in a list, these are the Indexs for each genotype
NUM_GENO = 4
COOP_SM = 0
COOP_LG = 1
SELF_SM = 2
SELF_LG = 3

#Some global lists for storing some data in to plot later.
data_c_s = list()
data_c_l = list()
data_s_s = list()
data_s_l = list()
large = list()
selfish = list()

## @brief Calculated the resources allocated to each genotype.
#  @param - the group to use
#  @retval - The resources allocated
#  Resources are allocated using the following formula
#  \f[
# r_i = \frac{ n_i . G_i . C_i }{\sum\limits_j (n_j . G_j . C_j )} . R
#  \f]
def Resource(group, R):

        #calculates the resource allocated to each genotype
        #calculate the sum part
        den = ( group[COOP_SM] * Gc * Cc ) + ( group[COOP_LG] * Gc * Cc ) + ( group[SELF_SM] * Gs * Cs ) +
        den = R / den  #and times it by the constant
        resources = [0] * NUM_GENO
        resources[COOP_SM] = den * group[COOP_SM] * Gc * Cc
        resources[COOP_LG] = den * group[COOP_LG] * Gc * Cc
        resources[SELF_SM] = den * group[SELF_SM] * Gs * Cs
        resources[SELF_LG] = den * group[SELF_LG] * Gs * Cs
        return resources


## @brief Calculate the growth of the population depending on the resource calculation
#  @param group - the group to use
#  @param resource - the resources consumed by the group
#  @retval - The resulting population
#  growth is calculated using the following formula
#  \f[
#       n_i (t + 1) = n_i (t) + \frac{r_i}{C_i} - K.n_i (t)
#  \f]
def GrowPopulation(group, resource):
        group[COOP_SM] = (group[COOP_SM] + ( resource[COOP_SM] / Cc ) - K * group[COOP_SM])
        group[COOP_LG] = (group[COOP_LG] + ( resource[COOP_LG] / Cc ) - K * group[COOP_LG])
        group[SELF_SM] = (group[SELF_SM] + ( resource[SELF_SM] / Cs ) - K * group[SELF_SM])
        group[SELF_LG] = (group[SELF_LG] + ( resource[SELF_LG] / Cs ) - K * group[SELF_LG])
        return group
## @brief Inialises the global lists and clears the output file
def InitWrite():
        f = open(outfile, 'w')
        f.write("COOP_SM,COOP_LG,SELF_SM,SELF_LG\n")
        f.close()
        data_c_s = list()
        data_c_l = list()
        data_s_s = list()
        data_s_l = list()
        large = list()
        selfish = list()

## @brief Writes the pool data to a text file and stores to list for plotting
#  @param pool - the pool to write
def WriteData(pool):
        f = open(outfile, 'a')
        f.write("%d,%d,%d,%d\n" % (pool[COOP_SM], pool[COOP_LG], pool[SELF_SM], pool[SELF_LG]))
        f.close()
        data_c_s.append(pool[COOP_SM] / float(N))
        data_c_l.append(pool[COOP_LG] / float(N))
        data_s_s.append(pool[SELF_SM] / float(N))
        data_s_l.append(pool[SELF_LG] / float(N))
        large.append((pool[SELF_LG] + pool[COOP_LG] )/ float(N))
        selfish.append((pool[SELF_LG] + pool[SELF_SM] )/ float(N))
        pass


## @brief plots the data in the global lists.
def PlotAll():
        pylab.figure(fig.number)
        pylab.xlabel("Generation")
        pylab.ylabel("Number of genotype")

        x = range(T)
        pylab.plot(x, data_c_s, 'b-', label="Co-op Small") # '.' is point, ',' is pixel
        pylab.plot(x, data_c_l, 'b:', label="Co-op Large") # '.' is point, ',' is pixel
        pylab.plot(x, data_s_s, 'r-', label="Selfish Small") # '.' is point, ',' is pixel
        pylab.plot(x, data_s_l, 'r:', label="Selfish Large") # '.' is point, ',' is pixel
        pylab.legend(loc='lower right')
        pylab.show()
        pylab.draw()

        #pylab.figure()
        pylab.xlabel("Generation")
        pylab.ylabel("Global frequency")
        pylab.plot(x, large, 'k:', label="Large Group Size")
        pylab.plot(x, selfish, 'k-', label="Selfish resource usage")
        pylab.legend(loc='upper right')
        pylab.show()
        pylab.draw()
        pass

## @brief some testing to check things work
def Test():
        test = [6.0,8.0,12.0,14.0]
        r = Resource(test, R_large)
        print ("Group :")
        print test
        print("Resources: ")
        print r
        GrowPopulation(test, r)
        print ("Group :")
        print test
        raw_input()
        pass
## @brief main function.
#  Executes the stages of the GA.
if "__main__" == __name__:
        #initialise an equally distributed population
        InitWrite()
        pool = list()
        for i in range(NUM_GENO):
                pool.append( float(N / NUM_GENO ) )
        print pool
        #WriteData(pool)
        #r = Resource(pool)
        #print r
        #pool = GrowPopulation(pool, r)
        #print pool
        for g in range(T):
                print("GENERATION %d" % g)
                WriteData(pool)
                #Group formation
                smallgroups = list()
                largegroups = list()
                #number of groups
                sm = int((pool[COOP_SM] + pool[SELF_SM]) / N_small)
                lg = int((pool[COOP_LG] + pool[SELF_LG]) / N_large)
                #calculate proportions
                if sm:#if we have any small groups to make
                        p_sm_coop = pool[COOP_SM] / ( pool[COOP_SM] + pool[SELF_SM])
                        for i in range(sm):
                                group = [0.0] * NUM_GENO
                                for i in range(N_small): #group size of n small
                                        if (random.random() < p_sm_coop):#choose a coop
                                                group[COOP_SM] += 1
                                        else:
                                                group[SELF_SM] += 1
                                smallgroups.append(group)
                if lg:#if we have any large groups to make
                        p_lg_coop = pool[COOP_LG] / ( pool[COOP_LG] + pool[SELF_LG])
                        for i in range(lg):
                                group = [0.0] * NUM_GENO
                                for i in range(N_large): #group size of n small
                                        if (random.random() < p_lg_coop):#choose a coop
                                                group[COOP_LG] += 1
                                        else:
                                                group[SELF_LG] += 1
                                largegroups.append(group)

                #Reproduction and resource allocation for allowed timesteps
                for group in largegroups:
                        for _t in range(t):
                                rl = Resource(group, R_large)
                                GrowPopulation(group, rl)
                for group in smallgroups:
                        for _t in range(t):
                                rs = Resource(group, R_small)
                                GrowPopulation(group, rs)
                #Migrant pool formation
                pool = [0.0]*NUM_GENO#reset pool - will remove any that didn't make it to groups
                for group in (largegroups + smallgroups):
                        for i in range(NUM_GENO):
                                pool[i] += group[i]
                print("Pool Size = %d" % sum(pool))
                #reduce pool
                scale = float(N) / float(sum(pool)) #scale so that have a population size of N
                print("Scale = %f" % scale)
                for i in range(NUM_GENO):
                        pool[i] = ((pool[i] * scale))
                print("Pool Size after scale = %d" % sum(pool))

        #end for T
        PlotAll()
        print("DONE!")
        raw_input()
        pass
```

Listing 1: Reimplementation of Powers et al. (2007)