Evolution of complexity
Assignment 2

Edward Kitcher
Student ID: 29845998
MSc Artificial Intelligence

January 8, 2019

Abstract

In this paper we will reproduce the results from Power's paper on the Individual Selection for Cooperative Group Formation [5] in which Powers demonstrates individuals which are small and cooperative become dominant, despite being at a disadvantage. We then further extend the experiment to allow for some variables to change over time. So we can test the hypothesis that time dependant variables which act on the environment will cause individuals to rise and fall periodically. From our new experiment we see that the small group under goes a sort of periodic behaviour, but ultimately small and cooperative becomes dominant.

1 Introduction

Power's paper sets out to show that conditions favouring either selfish or cooperative genotypes can arise by allowing individuals to change their environment, and therefore change the conditions which affect selection. This is known has niche construction [2], a fascinating area of evolutionary research. Previous works have shown that cooperative behaviours can be selected for, but only after externally imposing the conditions which favour cooperation.

Cooperative individuals can be thought of as those who act in a way which benefits the entire group, and not just themselves. These actions can

sometimes come at a cost to themselves, and therefore reduce their fitness. This lack of selfish behaviour seems contrary to the individuals self-interest as evolution acts upon individuals, not upon groups. And yet, there are plenty of real life examples of cooperative behaviour. Examples of this are bacteria which grow at a reduced rate in order to share a common resource more efficiently [4, 3, 1] which in turn, benefits the entire group. This behaviour of individuals sacrificing immediate reward for a delayed one appears across various research disciplines. However, there is a benefit to being selfish in a group of co-operators. In the context of the bacteria example a selfish individual could consume a resource as much as they like and therefore grow at a faster rate at the expense of the cooperators in the group.

1.1 Model description

In Power's model individuals are either cooperative or selfish and belong to either large or small groups. This creates a total of 4 possible genotypes. Selfish individuals are at an advantage as they have a higher growth rate compared to cooperative ($G_s > G_c$). Larger groups are also at an advantage as they receive a higher consumption rate. All the parameters of the model can be seen below

Parameter	Value
Growth rate (cooperative), G_c	0.018
Growth rate (selfish), G_s	0.02
Consumption rate (cooperative), C_c	0.1
Consumption rate (selfish), C_s	0.2
Resource Influx (small), R small	4
Resource Influx (large) R large	50
Initial Group Size (small), S_s	4
Initial Group size (large), S_l	40
Death Rate, K	0.1
Population size, N	4000
Number of generations, T	120
Reproduction time-steps, t	4

The 4 variations are initially created will equal population sizes which then under go reproduction within groups for t time steps. Groups are then

dispersed and then rescaled back to size N, so as to retain the same proportions of the given genotype. The reproduction, dispersal and reformation is then repeated over the given number of generations, T.

2 Reimplementation Results

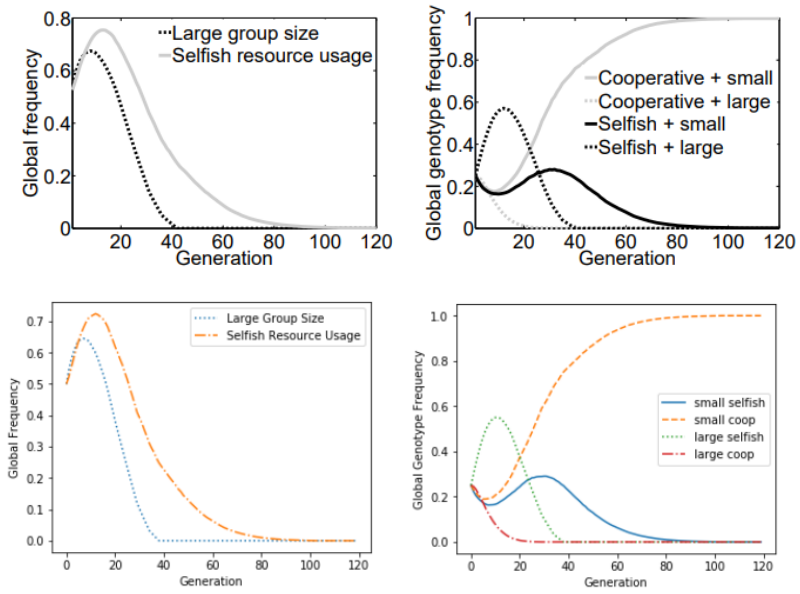

Figure 1: Power's results (above) along with my reimplemented results (below). LHS plots: Average enviroment and strategy through time. RHS plot: change in genotype frequencies over time.

We can clearly see from figure 1 that my results match with Power's. The LHS plots allow us to see how the alleles for large groups size and for selfish grow fast initially but then quickly crash down. This can be seen by looking at the RHS, where we see the initial rise in large selfish is due to immediate fall of the large cooperative genotype. So large groups become favourable for selfish individuals and small groups for cooperative. But once all the cooperators in the large group have been exploited the selfish individuals in these groups start to drop. This causes small selfish to rise momentary

as they receive a greater proportion of the total population, but due to the constant dispersal of small groups they are not able to grow quickly enough and eventually become out competed by small-cooperative.

So even though small-cooperative was initially selected against, it managed to become dominant through the natural changes in the environment.

2.1 Method

We originally represented groups as bit strings consisting of either 0's (selfish) or 1's (cooperative), E.g. [1,0,1,0]. Each size category (small/large) was a list containing a certain number of groups. However, we decided to simplify things by represent groups as a list with only 2 elements instead, each element corresponds to the number of individuals of a certain allele. E.g. [No. of selfish, No. of coop]. This turned out to be sufficient. A crucial part of the implementation is to create groups without replacement.

3 Extension Description

For the extension we are interested in adapting the model to investigating how individuals change their environment as the environment itself is changing (with respect to time). In the original model, individuals change their environment via niche construction. However, to make the model more realistic, we attempt to add in the effects of seasonal change on some of the model variables. Specifically, death rate (K), time spent reproducing (t) and group size.
The effect of the seasons is of course dependant on individual species. For some species, a particular season might create periods of growth, while another species will experience periods of decline. For simplicity, our model will represent the seasonal changes by using a Sine wave, starting with similar initial conditions as before. The model begins in 'Spring' (I.e. paraments remain unchanged), leading to Summer (where the variables will be at their maximum), continuing into Autumn (variables will start to approach the initial conditions) and finally into Winter (where the variables will be at their minimum) before returning to spring and repeating.

Our hypothesis is; time dependant variables which act on the environment will cause individuals to rise and fall periodically, without one ever becoming completely dominant, as the environment will create conditions which favour various genotypes.
Selfish individuals do well when there are plenty of cooperatives to exploit, so as the model progresses into summer the group sizes will increase which will mean they are more likely to enter a group with cooperatives. Conversely, during Winter the group sizes will be smaller and so cooperatives will be able to regain their lost numbers. As the cycle approaches spring we expect the results will repeat themselves.

Variables K, t and Groupsize are now calculated by the following equations.
K: Ranging from 0 to 0.2 (beginning at 0.1)

$$K(T) = 0.1(\sin(T) + 1) \tag{1}$$

t: Ranging from 0 to 4 (beginning at 2)

$$t(T) = 2 + 2\sin(T) \tag{2}$$

Groupsize(T): Small groups (S_s) sizes range from 2 to 6 (beginning at 4). Large group sizes (S_l) range from 20 to 60 (beginning at 40).

$$Groupsize(T) = S_j + (\frac{S_j}{2})\sin(T) \tag{3}$$

Note: while we use the current generation number (T) as the dependant variable in the sine wave, the value is converted into radians, e.g generation 90 will be $\pi/2$.

4 Extension Results

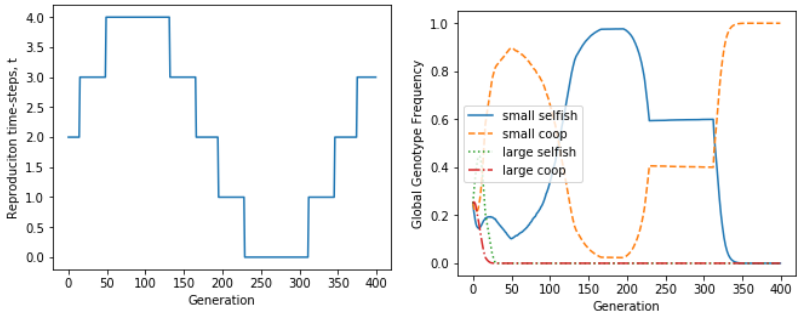

Figure 2: LHS: Reproduction time steps changing as generations increase. This graph will enable readers to easily see how the sine wave is changing over T. RHS Change in genotype frequencies over time

We can see that the model starts out by replicating the same results as before for generations up to about 50. Shortly after, as generation approaches the variable maximums (generation 90), the selfish genotype starts to rise due to the increased group size and the increased reproduction time steps. Now the small-selfish have a greater chance of entering a group of cooperatives as well as more time to exploit them. This causes a decrease in the number of small-cooperatives as they become less represented in the population. But now as the variables are approaching their initial values we see the small-cooperatives making a comeback as the variables start to go in their favour again. The growth of populations freezes during winter due to the value of t being rounded to zero. Once t starts to rise again we see small-cooperative grow at the cost of small-selfish due to the favourable conditions (smaller group sizes and small t). Until eventually small cooperative becomes dominant.
However, while this does seem to support Power's results the model is not sophisticated enough to truly test the hypothesis. It remains unclear whether individuals will rise and fall periodically given proper values for the variables. Also, there is a mistake in the way death rate, K has been calculated, it should be shifted by π to correspond to the correct season.

5 Conclusions and Evaluation

While the extension does seem to support Power's results, the detail and sophistication of the model isn't sufficient to be used as evidence. Also, the choices of the equations are not justified appropriately with support from relevant literature. The discussion is too brief as a result as not much can be extrapolated from the results without appropriate context.

References

[1] J.-U. Kreft. Biofilms promote altruism. *Microbiology*, 150(8):2751–2760, 2004.

[2] F. J. Odling-Smee, K. N. Laland, and M. W. Feldman. *Niche construction: the neglected process in evolution*. Number 37. Princeton university press, 2003.

[3] T. Pfeiffer and S. Bonhoeffer. An evolutionary scenario for the transition to undifferentiated multicellularity. *Proceedings of the National Academy of Sciences*, 100(3):1095–1098, 2003.

[4] T. Pfeiffer, S. Schuster, and S. Bonhoeffer. Cooperation and competition in the evolution of atp-producing pathways. *Science*, 292(5516):504–507, 2001.

[5] S. T. Powers, A. S. Penn, and R. A. Watson. Individual selection for cooperative group formation. In *European Conference on Artificial Life*, pages 585–594. Springer, 2007.

6 Appendix

6.1 Code for reimplementation

Listing 1: Code for reimplementation

```
import numpy as np
import random
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
```

```
smallsize = 4
largesize = 40

Gc = 0.018 # co op growth rate
Gs = 0.02 # selfish growth rate

Cc = 0.1 # co op resource consumption rate
Cs = 0.2 # selfish resource consumption rate

N = 4000 #pop number
K = 0.1 # death rate

R_small = 4 # resource influx, R.
R_large = 50 #resource influx, R.

generations = 120 #number of generations

small = [N/4, N/4, 4] # small = [selfish, coop, group size]
    TOTAL NUMBER OF GENOTYPES IN SMALL
large = [N/4, N/4, 40] # large = [selfish, coop, group size]
    TOTAL NUMBER OF GENOTYPES IN lARGE

def creategroups(small_or_large): # creates a list of lists,
    each sublist is a group with [No. of selfish,No. of coop]
     collection = []
     total = small_or_large[0] + small_or_large[1] #total number
         of that genotype
     replace = total
     A = small_or_large[0]
     for i in range(0, int(total/small_or_large[2])): # total
         number of groups for that size which will be created
          group=[]
          selfish_count = 0
          coop_count = 0
          for j in range(0,small_or_large[2]):
               if random.random() < A/(replace) : # number of
                   selfish, so A = small[0]
                    selfish_count += 1
                    A -= 1
                    replace -= 1
               else:
                    coop_count += 1
                    replace -= 1
          collection += [[selfish_count,coop_count]]
```

```
    return collection # returns a list of lists, each sub list
        is a group

def reproduction_small(small):
    new_group = []
    R_influx = R_small
    for i in range(0,len(small)): # so will be 500 at first

        group_i = small[i]

        n_selfish = group_i[0]

        top_selfish = n_selfish*Gs*Cs # top part of r, for
            selfish

        n_coop = group_i[1] # as co op = 1, sum will tell us how
             many there are
        top_coop = n_coop*Gc*Cc # top part of r, for co op



        bottom_both = (top_selfish + top_coop) # bottom part of
            the fraction

        r_selfish = (top_selfish/bottom_both)*R_small # THIS
            WILL GIVE ME THE r VALUE FOR THAT GROUP
        r_coop = (top_coop/bottom_both)*R_small

        #DONT TAKE INTIGER UNLESS ON LAST STEP
        n_selfish = (n_selfish + (r_selfish/Cs) - K*n_selfish) #
             number of selfish now in group
        n_coop = (n_coop + (r_coop/Cc) - K*n_coop)

        group_i=[n_selfish,n_coop]
        new_group += [group_i]


    return new_group

def reproduction_large(large):

    new_group = []
    R_influx = R_large

    for i in range(0,len(large)):
```

```
        group_i = large[i]

        n_selfish = group_i[0]
        top_selfish = n_selfish*Gs*Cs

        n_coop = group_i[1] # as coop = 1, sum will tell us how
            many there are
        top_coop = n_coop*Gc*Cc

        bottom_both = (top_selfish + top_coop)

        r_selfish = (top_selfish/bottom_both)*R_large # THIS
            WILL GIVE ME THE r VALUE FOR THAT GROUP
        r_coop = (top_coop/bottom_both)*R_large

        n_selfish = (n_selfish + (r_selfish/Cs) - K*n_selfish) #
            number of selfish now in group
        n_coop = (n_coop + (r_coop/Cc) - K*n_coop)

        group_i=[n_selfish,n_coop]
        new_group += [group_i]

    return new_group

def rescale(small_pool,large_pool): # rescales the migrant pool
    into new small and large (sum ~= 4000)
    pool_size = small_pool[0] + small_pool[1] + large_pool[0] +
        large_pool[1]

    new_small_selfish = int(N*small_pool[0]/pool_size )

    new_small_coop = int(N*small_pool[1]/pool_size )

    new_large_selfish = int(N*large_pool[0]/pool_size)

    new_large_coop = int(N*large_pool[1]/pool_size)

    small = [new_small_selfish, new_small_coop, smallsize]
    large = [new_large_selfish, new_large_coop, largesize]
#     print(small[0] + small[1] + large[0] + large[1])
    return small,large
# 1. Initialisation: Initialise the migrant pool with N
    individuals.

#first time step there are equal in each variation
```

```
store1 = []
store2 = []
store3 = []
store4 = []
store5 = []
store6 = []
gen = 120
R_small = 4 # resource influx, R.
R_large = 50 #resource influx, R.
small = [N/4, N/4, 4] # small = [selfish (0), coop (1)] TOTAL
    NUMBER OF GENOTYPES IN SMALL
large = [N/4, N/4, 40] # large = [selfish, coop] TOTAL NUMBER OF
     GENOTYPES IN lARGE
print('original_small', small,'original_large', large)

# THIS IS WHERE T LOOP BEGINS

# for T in range(0,generations):
for T in range(0,gen):
     store1 += [small[0]/4000] #used for plots
     store2 += [small[1]/4000]
     store3 += [large[0]/4000]
     store4 += [large[1]/4000]
     store5 += [(large[0]+large[1])/4000] # both large genotypes
     store6 += [(small[0]+large[0])/4000] # both selfish
         genotypes
     totalselfishA = 0
     totalcoopA = 0
     totalselfishB = 0
     totalcoopB = 0
# 2. Group formation (aggregation): Assign individuals in the
    migrant pool to groups, as described in the main text below.

     A = creategroups(small)
     B = creategroups(large)

# 3. Reproduction: Perform reproduction within groups for t time
    -steps, as described in the text above
      for t in range(0,4):
          A = reproduction_small(A)
          B = reproduction_large(B)

# 4. Migrant pool formation (dispersal): Return the progeny of
    each group to the migrant pool
      for i in range(0,len(A)):
```

```
        totalselfishA += A[i][0]
        totalcoopA += A[i][1]
    for i in range(0,len(B)):
        totalselfishB += B[i][0]
        totalcoopB += B[i][1]

    small_pool = [ int(totalselfishA) , int(totalcoopA) ] #
        selfish (0), coop (1)
    large_pool = [ int(totalselfishB) , int(totalcoopB) ] #
        selfish (0), coop (1)
#     print('small pool:', small_pool, 'large pool:',large_pool)

# 5. Maintaining the global carrying capacity:
# Rescale the migrant pool back to size N, retaining the
    proportion of individuals with each genotype.
    small = rescale(small_pool,large_pool)[0]
    large = rescale(small_pool,large_pool)[1]
#     print('the new small',small ,'the new large',large)

figure(figsize=(5,4))
plt.xlabel('Generation')
plt.ylabel('Global Genotype Frequency')
plt.plot(range(0,gen), store1, label='small selfish')
plt.plot(range(0,gen), store2, label='small coop', linestyle='--
    ')
plt.plot(range(0,gen), store3, label='large selfish', linestyle
    =':')
plt.plot(range(0,gen), store4, label='large coop', linestyle = '
    -.')
plt.legend()
plt.show

figure(figsize=(5,4))
plt.xlabel('Generation')
plt.ylabel('Global Frequency')
plt.plot(range(0,gen), store5, label='Large Group Size',
    linestyle =':')
plt.plot(range(0,gen), store6, label='Selfish Resource Usage',
    linestyle = '-.')
plt.legend()
plt.show
```

6.2 Code for Extension

Listing 2: Code for extension

```
import numpy as np
import random
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
import math
from numpy import arange, sin, pi

def reproduction_range(T):
    t = 2 + (2*math.sin(math.radians(T))) # + 1 to stop negative
        values
    return round(t)

def K_e(T):
    a = K*(math.sin(math.radians(T)) + 1)
    return a

def reproduction_small_E(small,T): # reliant on K_e(T)
    new_group = []
    R_influx = R_small
    for i in range(0,len(small)): # so will be 500 at first

        group_i = small[i]

        n_selfish = group_i[0]

        top_selfish = n_selfish*Gs*Cs # top part of r, for
            selfish

        n_coop = group_i[1] # as co op = 1, sum will tell us how
             many there are
        top_coop = n_coop*Gc*Cc # top part of r, for co op



        bottom_both = (top_selfish + top_coop) # bottom part of
            the fraction

        r_selfish = (top_selfish/bottom_both)*R_small # THIS
            WILL GIVE ME THE r VALUE FOR THAT GROUP
        r_coop = (top_coop/bottom_both)*R_small

        #DONT TAKE INTIGER UNLESS ON LAST STEP
        n_selfish = (n_selfish + (r_selfish/Cs) - K_e(T)*
            n_selfish) # number of selfish now in group
```

```
            n_coop = (n_coop + (r_coop/Cc) - K_e(T)*n_coop)

            group_i=[n_selfish,n_coop]
            new_group += [group_i]


        return new_group

def reproduction_large_E(large,T): # death rate is affected
        new_group = []
        R_influx = R_large

        for i in range(0,len(large)):
            group_i = large[i]

            n_selfish = group_i[0]
            top_selfish = n_selfish*Gs*Cs

            n_coop = group_i[1] # as co op = 1, sum will tell us how
                 many there are
            top_coop = n_coop*Gc*Cc

            bottom_both = (top_selfish + top_coop)

            r_selfish = (top_selfish/bottom_both)*R_large # THIS
                WILL GIVE ME THE r VALUE FOR THAT GROUP
            r_coop = (top_coop/bottom_both)*R_large

            n_selfish = (n_selfish + (r_selfish/Cs) - K_e(T)*
                n_selfish) # number of selfish now in group
            n_coop = (n_coop + (r_coop/Cc) - K_e(T)*n_coop)

            group_i=[n_selfish,n_coop]
            new_group += [group_i]

        return new_group

smallsize = 4
def smallsize_E(x): # adds +2 and -2
        a = smallsize + (smallsize/2)*(math.sin(math.radians(x))) #
            (math.sin(math.radians(x)))
        return round(a)

largesize = 40
def largesize_E(x):
```

```
    a = largesize + (   (largesize/2)*(math.sin(math.radians(x)))
        )  # (math.sin(math.radians(x)) + 1)
    return round(a)

# 1. Initialisation: Initialise the migrant pool with N
    individuals.

#first time step there are equal in each variation
store1 = []
store2 = []
store3 = []
store4 = []
store5 = []
store6 = []
store7 = []
store8 = []
store9 = []
store10 = []
gen = 400
R_small = 4 # resource influx, R.
R_large = 50 #resource influx, R.
small = [N/4, N/4, smallsize] # small = [selfish (0), coop (1)]
    TOTAL NUMBER OF GENOTYPES IN SMALL
large = [N/4, N/4, largesize] # large = [selfish, coop] TOTAL
    NUMBER OF GENOTYPES IN lARGE
print('original_small', small,'original_large', large)

for counter in range(0,gen):
    T = counter
    store1 += [small[0]/4000] #used for plots
    store2 += [small[1]/4000]
    store3 += [large[0]/4000]
    store4 += [large[1]/4000]
    store5 += [(large[0]+large[1])/4000]
    store6 += [(small[0]+large[0])/4000]
    store7 += [reproduction_range(T)]
    store8 += [K_e(T)]
    store9 += [smallsize_E(T)]
    store10 += [largesize_E(T)]
    totalselfishA = 0
    totalcoopA = 0
    totalselfishB = 0
    totalcoopB = 0
# 2. Group formation (aggregation): Assign individuals in the
    migrant pool to groups, as described in the main text below.
```

```
    A = creategroups(small)
    B = creategroups(large)
# 3. Reproduction: Perform reproduction within groups for t time
    -steps, as described in the text above
    for t in range(0,reproduction_range(T)):
        A = reproduction_small_E(A,T)
        B = reproduction_large_E(B,T)

# 4. Migrant pool formation (dispersal): Return the progeny of
    each group to the migrant pool
    for i in range(0,len(A)):
        totalselfishA += A[i][0]
        totalcoopA += A[i][1]
    for i in range(0,len(B)):
        totalselfishB += B[i][0]
        totalcoopB += B[i][1]

    small_pool = [ int(totalselfishA) , int(totalcoopA) ] #
        selfish (0), coop (1)
    large_pool = [ int(totalselfishB) , int(totalcoopB) ] #
        selfish (0), coop (1)
#     print('small pool:', small_pool, 'large pool:',large_pool)

# 5. Maintaining the global carrying capacity:
# Rescale the migrant pool back to size N, retaining the
    proportion of individuals with each genotype.
    small = rescale_E(small_pool,large_pool,T)[0]
    large = rescale_E(small_pool,large_pool,T)[1]

figure(figsize=(5,4))
plt.xlabel('Generation')
plt.ylabel('Global Genotype Frequency')
plt.plot(range(0,gen), store1, label='small selfish')
plt.plot(range(0,gen), store2, label='small coop', linestyle='--
    ')
plt.plot(range(0,gen), store3, label='large selfish', linestyle
    =':')
plt.plot(range(0,gen), store4, label='large coop', linestyle = '
    -.')
plt.legend()
plt.show


figure(figsize=(5,4))
```

```
plt.xlabel('Generation')
plt.ylabel('Reproduciton time-steps, t')
plt.plot(range(0,gen), store7)
plt.show
```