Evolution of Complexity Assignment 2

Group Selection

Zhihan Wang

[1] Data Science
[2] ECS
[3] University of Southampton
[4] zw3u18@soton.ac.uk

1 Introduction

This paper firstly describes the original paper[1] in section 1 and then reimplements and analyses the results from it. Extension based on the knowledge in the original paper is covered in the last section.

Group selection could be defined as a mechanism of evolution in which natural selection acts at the level of groups instead of individuals[3]. It is very common in the nature, both in the same species and between different species. Selection not only happens between groups but also within each group. Within each group, selfish cheaters tend to have a higher fitness as they exploit more resources with less pay-off[4]. But cooperation is always preferred between groups. The end results of the group selection in a population would consider these both two factors. However, cooperation is more common in the nature even not as much fit as the selfish behaviour within a group. Therefore, most past theories about group selection are based on an assumption that the evolving environment is much better for cooperation to fit. In these results, cooperation wins at last. But because of postulating external environmental conditions, these models lose the power of explanation when there are no external conditions imposed[1].

1.1 Brief description

In the paper[1], instead of evolving under the conditions favouring cooperation, individuals adapt the environment by themselves. It is an individual adaptation model, also known as niche construction, in which which strategy is preferred by individual adaptation is influenced by the environmental condition. At the same time, which environmental condition would evolve is affected by the strategies played in the groups.

An example is the aggregation and dispersal model, in which each individual comes with two traits, the strategy it plays and the size of group it joins. To be more explicit, individuals could choose to be cooperators or cheaters and choose large or small groups to join. A group size(environmental condition) and a strategy can be regarded as a combination to evolve together. Therefore, there are no extra external conditions imposed.

Two main conclusions are acquired from this model. The first one is in the pairwise comparisons, individuals with a strategy in large size groups always outcompete those in small size groups owing to more resource assigned to larger groups. And selfish strategy always outcompetes cooperative one in the same groups as the selection favouring selfish cheaters within groups. However, this outcome does not happen in the situation with all four combinations in equal proportion in a population. So another conclusion is that in the population with all four combinations, the cooperative strategy with a small group size wins at last. Though selfish cheaters outcompete cooperators within groups, cooperators are favoured between groups. By reducing group size, increasing sample errors between groups could counter the lost frequency of cooperators within groups so as to generate the final result.

1.2 Details

There are four combinations to evolve. One part of each combination is the strategy played by an individual, and the other is the size of group it joins. Therefore, these four combinations are cooperative+small, cooperative+large, selfish+small, selfish+large respectively.

Each individual is not explicitly represented like a real genotype. As said before, it comes with two traits, the strategy and the group size. Individuals evolving in this model are matched with the frequency of each combination.

There is no defined fitness function as well. The combination with increasing global frequency would be the one with the highest fitness.

The overall algorithm of the aggregation and dispersal model could be simply presented as below:

Initialisation : Initialise the migrant pool with N individuals.

Aggregation : Assign N individuals to groups.

Reproduction : Individuals within groups reproduce t timesteps.

Dispersal : Return the progeny of each group to the original pool.

Scale : Rescale the pool to size N according to the proportion of each genotype in the last step.

Iteration : Repeat from Aggregation step.

According to this algorithm, a steady state is applied in the dispersal step, since the progeny is returned to the original pool mixing with parents. Besides, in the scale step, rescaling the pool with the proportion from last step results could be regarded as a roulette wheel in some extent.

2 Reimplementation

In order to reproduce, genotpye i requires a share of group resource. The resource it received, r_i, is defined in equation 1.

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

n_i is the number of copies of genotype i, G_i is its growth rate and C_i is its resource consumption rate which could be regarded as the amount of resource required for one individual to make one clone. R is a share of the group's resource influx. It is a constant for each individual in the same group but different between groups. A noteworthy point is that R is related to the group size. A larger R is assigned to a larger group size.

With received resource, individuals begin to reproduce. The number of individuals after reproduction is defined in equation 2. t is the timestep and K is global death rate same for everyone in the population.

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

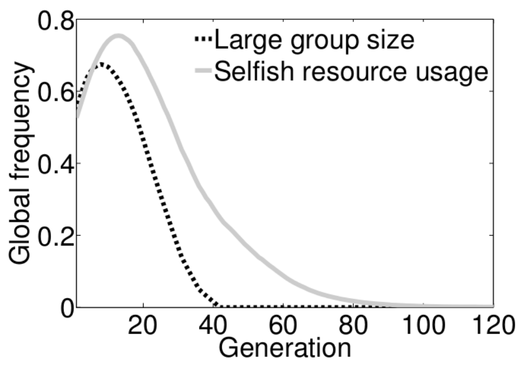
To reimplement, detailed parameters are presented in the table below.

Table 1: Parameters used in the reimplementation.

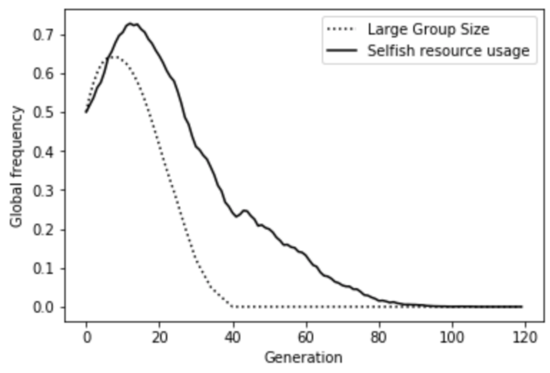
Parameter	Value
Growth rate(cooperative), G_c	0.018
Growth rate(selfish), G_s	0.02
Consumption rate(cooperative), C_c	0.1
Consumption rate(slefish), C_s	0.2
Population size, N	4000
Number of iterations, T	120
Small group size, N_{small}	4
Large group size, N_{large}	40
Number of timesteps, t	4
R for small group size, R_{small}	4
R for large group size, R_{large}	50
Death rate, K	0.1

2.1 Figure comparison

In this section, I am going to show the reimplemented results comparing with the original ones. From the figures, we can see the reproduced ones are very close to the original ones. Fig. 1 shows how the global frequency of the large group and selfish cheaters change over time. From this figure, both large groups and selfish cheaters are favoured at the beginning but after around 20 iterations, the fitness of these two features begin to decline. Finally, they are both removed from the population. What's more, it takes more time to remove the selfish cheaters from the population than the large groups.



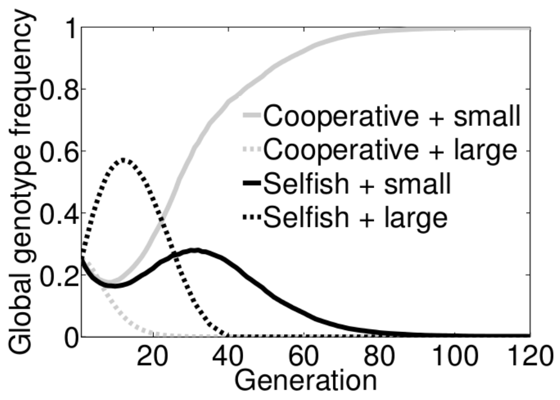
(a) Original plot



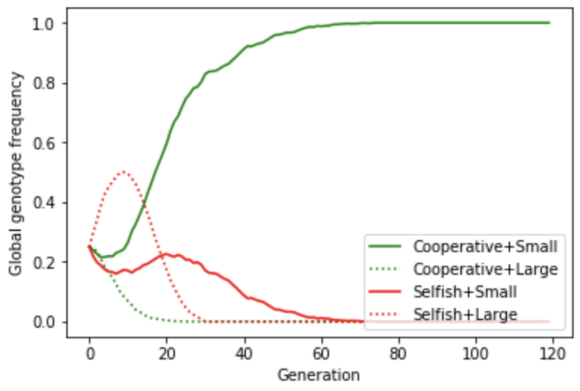
(b) Reimplemented plot

Fig. 1: Average environment and strategy through time

For each genotype, it could be a cooperator or a selfish cheater, and it could be in a large group or a small group. Fig. 2 shows the change of frequency of each genotype. At the very beginning, selfish cheaters in the large groups have the highest global fitness. But with the frequency of cheaters rapidly increasing, cooperators are extinct due to limited resources. This circumstance is followed with a sudden drop of the frequency of cheaters. The result can be explained that cheaters with an inefficient resource usage strategy cannot benefit from a group without cooperators anymore. So large groups are extinct finally. Compared with large groups, small groups are in a lower fitness at first with a slight decrease. This is due to the resource bonus from larger groups, in which individuals can receive extra resource influx.Therefore, with more resources, individuals in larger groups are able to reproduce more efficiently at the beginning. As said in the section 1, small groups favour cooperators due to the combined results from selection acting within groups and between groups, so the frequency of cooperators is higher than cheaters after dispersal and aggregation process. But an another noteworthy point is the frequency of



(a) Original plot



(b) Reimplemented plot

Fig. 2: Change in genotype frequencies over time

cheaters in small groups slightly increases with the rise of the frequency of cooperators. This outcome can be explained that within groups, cheaters benefit from the situation with more cooperators. So there is a short-term rising in the frequency. However, due to the constant dispersal and aggregation which prefer cooperation, cheaters are extinct finally as well and cooperators in small groups reach the fixation.

3 Extension

The above discussions about the original paper focus on two types of groups, small size and large size. However, in the real nature, it is naive to simply divide all groups to two types. The structure in the nature is far more complex. What if there is a type of group in the middle? More explicitly, if individuals have an another group choice, medium group, which combination will evolve finally? And obviously, with more group size choices, this model would be more reasonable as well.

In this section, I am going to introduce a new group type, medium group, into the individual adaptation model discussed above to see what will happen. Since small groups favour cooperation, whereas large groups prefer cheaters, which strategy will win in a medium group? And how does the composition of a population change with the new type of groups involved? In the previous discussions, the combination of cooperation with small groups outcompetes all the others and becomes the only winner in the population. But the outcome like this is very strange in the real nature since there is no population, in which all members are in the same strategy. So I conjecture individuals with different strategies might reach a balance and coexist in the population with more group types.

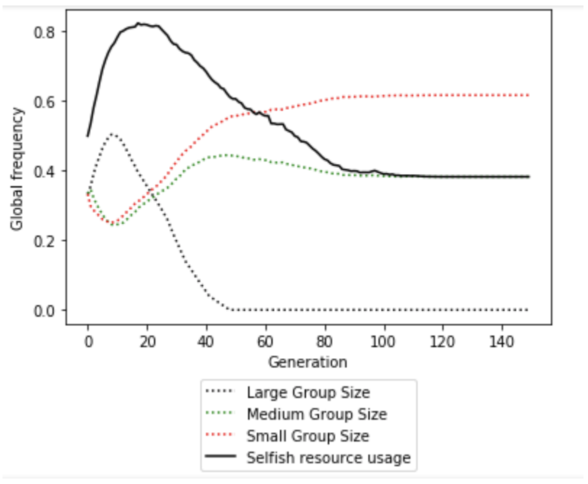
The size of medium group is set up to be the mean of the value of small group(N=4) and the value of large group(N=40). The influx R for medium group is also the mean of values of small and large groups. The parameters in table 1 are still used in this part except the number of iteration T and population size N. To ensure the results reaching a fixation in the experiment, an increased number of iteration replaces the original one. Besides, in order to set up the initial number of individuals to integer, a new N is used here. The additional parameters are implemented in the table below.

Table 2: Additional parameters used in extension.

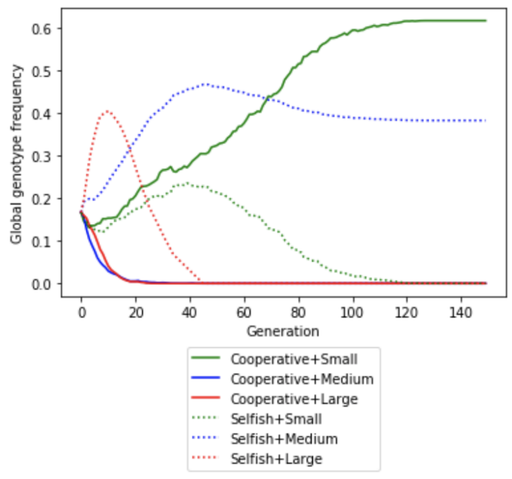
Parameter	Value
Number of Iteration, T'	150
medium group size, N_{medium}	22
Population size, N'	6000
R for medium group size, R_{medium}	27

3.1 Results

The results in a population with three group sizes are shown in Fig. 3. Similar to the above figures, the left-hand plot shows the average change of global frequency of three types of groups and selfish strategy usage over time. After a long run, large groups die off, but medium groups and small groups coexist in one population. From this plot, we can also see that the selfish usage reaches a fixation between zero and one, which means selfish cheaters and cooperators exactly coexist in this population as well.



(a) Average change



(b) Individual frequency

Fig. 3: Left-hand plot is the average change of three environmental conditions and strategy usage over time. Right-hand plot is the change of frequency of each genotype through time

More details about each genotype can be seen in the right-hand plot. The most obvious point is that cooperators in small groups and cheaters in medium groups outcompete others at last and arrive at a balance. The cooperators in the large groups and the cheaters in both small and large groups have similar patterns with conclusions in[1]. But it takes cooperators in the small groups more time to evolve in this more complex model. The reason why the frequency of cheaters in medium groups grow faster than in small groups is that larger size groups have a resource bonus. Another point is that cooperators in the medium groups have similar performance with those in large groups, extinct within twenty iterations.

3.2 Conclusion

With introduction of medium groups, the complexity of the model has been increased and the new model becomes more reasonable considering nature reality. From the extension sector, medium groups favour cheaters. This result can be explained that comparing with small groups, medium groups with more resources promote the growth of selfish cheaters and therefore force cooperators in these groups to be extinct. It is very similar to the performance of the large groups in the previous discussions. But the difference is that selfish behaviours in the medium groups are kept at the end. In addition, cooperators are in a larger proportion in this new model, able to coexist with selfish cheaters. This is an interesting and realistic point, more advanced than the original one.

However, the number of group types in this model are still insufficient. In other words, in the real nature, the groups in a population should not be discrete. Therefore, in the future work, a group type should not just be limited in one specific value. For example, small size groups maybe include groups with the number of members between one and four, instead of simply assigned with a value, 4. And medium groups might be assigned values between a and b, rather than a fixed value, 22. This kind of continuous setting would be followed with an another question. That is how to define 'large', 'medium' and 'small'. In other words, what are the borders between groups and how to find them? More future works are required to implement on these questions.

References

1. Powers, Simon T., Alexandra S. Penn, and Richard A. Watson. "Individual selection for cooperative group formation." European Conference on Artificial Life. Springer, Berlin, Heidelberg, 2007.
2. Wilson, David Sloan. "A theory of group selection." Proceedings of the national academy of sciences 72.1 (1975): 143-146.
3. Williams, George C. Group selection. Routledge, 2017.
4. Goddard, Matthew R., and Austin Burt. "Recurrent invasion and extinction of a selfish gene." Proceedings of the National Academy of Sciences 96.24 (1999): 13880-13885.
5. Powers, Simon T. Social niche construction: evolutionary explanations for cooperative group formation. Diss. University of Southampton, 2010.

A reimplement code

~~~
```python
import math
import random
import pylab
outfile = "pool.txt"
fig = pylab.figure()
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 = 0.01 ## Growth rate for a cooperator
Gs = 0.02 ## Growth rate for selfish
Cc = 0.2 ## 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
```
~~~

```
    <Figure size 432x288 with 0 Axes>
```

~~~
```python
Comb_num = 4
COOP_SM = 0
COOP_LG = 1
SELF_SM = 2
SELF_LG = 3
```
~~~

~~~
```python
fre_cs = list()
fre_cl = list()
fre_ss = list()
fre_sl = list()
large = list()
selfish = list()
small = list()
coop= list()
```
~~~

~~~
```python
def Resource(group, R):
    addall = ( group[COOP_SM] * Gc * Cc ) + ( group[COOP_LG] * Gc * Cc ) + ( group[SELF_SM
    addall = R / addall
    resources = [0] * Comb_num
    resources[COOP_SM] = addall * group[COOP_SM] * Gc * Cc
    resources[COOP_LG] = addall * group[COOP_LG] * Gc * Cc
    resources[SELF_SM] = addall * group[SELF_SM] * Gs * Cs
~~~

```
    resources[SELF_LG] = addall * group[SELF_LG] * Gs * Cs
    return resources
```

```python
def Population(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
```

```python
def InitWrite():
    f = open(outfile, 'w')
    f.write("COOP_SM,COOP_LG,SELF_SM,SELF_LG\n")
    f.close()
    fre_cs = list()
    fre_cl = list()
    fre_ss = list()
    fre_sl = list()
    large = list()
    selfish = list()
    small = list()
    coop = list()
```

```python
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_
    f.close()
    fre_cs.append(pool[COOP_SM] / float(N))
    fre_cl.append(pool[COOP_LG] / float(N))
    fre_ss.append(pool[SELF_SM] / float(N))
    fre_sl.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))
    small.append((pool[SELF_SM] + pool[COOP_SM] )/ float(N))
    coop.append((pool[COOP_LG] + pool[COOP_SM] )/ float(N))
    pass

```

```python
def PlotAll():
    pylab.figure(fig.number)
    pylab.xlabel("Generation")
    pylab.ylabel("Global frequency")
    x=range(T)
    pylab.plot(x, large, 'k:', label="Large Group Size")
    pylab.plot(x, selfish, 'k-', label="Selfish resource usage")
    pylab.plot(x, small, 'r:', label="Small Group Size")
    pylab.plot(x, coop, 'r-', label="Cooperative resource usage")
    pylab.legend(loc='upper right')
    pylab.show()
    pylab.draw()
    pylab.xlabel("Generation")
    pylab.ylabel("Global genotype frequency")
    pylab.plot(x, fre_cs, 'g-', label="Cooperative+Small")
    pylab.plot(x, fre_cl, 'g:', label="Cooperative+Large")
    pylab.plot(x, fre_ss, 'r-', label="Selfish+Small")
    pylab.plot(x, fre_sl, 'r:', label="Selfish+Large")
    pylab.legend(loc='lower right')
    pylab.show()
    pylab.draw()
    pass
```

```python
def Test():
    test = [6.0,8.0,12.0,14.0]
    r = Resource(test, R_large)
    print ("Group :")
    print (test)
    print("Resources: ")
    print (r)
    Population(test, r)
    print ("Group :")
    print (test)
    input()
    pass
```

```python
Test()
```

```
    Group :
    [6.0, 8.0, 12.0, 14.0]
    Resources:
    [2.542372881355932, 3.3898305084745766, 20.338983050847457, 23.728813559322035]
    Group :
    [30.823728813559317, 41.09830508474577, 112.49491525423727, 131.24406779661015]
```

```python
if "__main__" == __name__:
    InitWrite()
    pool = list()
    for i in range(Comb_num):
        pool.append( float(N / Comb_num ) )
    print (pool)
    for g in range(T):
        print("GENERATION %d" % g)
        WriteData(pool)
        smallgroups = list()
        largegroups = list()
        sm = int((pool[COOP_SM] + pool[SELF_SM]) / N_small)
        md = int((pool[COOP_SM] + pool[SELF_SM]) / N_small)
        lg = int((pool[COOP_LG] + pool[SELF_LG]) / N_large)
        if sm:
            p_sm_coop = pool[COOP_SM] / ( pool[COOP_SM] + pool[SELF_SM])
            for i in range(sm):
                group = [0.0] * Comb_num
                for i in range(N_small):
                    if (random.random() < p_sm_coop):
                        group[COOP_SM] += 1
                    else:
                        group[SELF_SM] += 1
                smallgroups.append(group)
        if lg:
            p_lg_coop = pool[COOP_LG] / ( pool[COOP_LG] + pool[SELF_LG])
            for i in range(lg):
                group = [0.0] * Comb_num
                for i in range(N_large):
                    if (random.random() < p_lg_coop):
                        group[COOP_LG] += 1
                    else:
                        group[SELF_LG] += 1
                largegroups.append(group)

        for group in largegroups:
            for _t in range(t):
                rl = Resource(group, R_large)
                Population(group, rl)
        for group in smallgroups:
            for _t in range(t):
                rs = Resource(group, R_small)
                Population(group, rs)
        pool = [0.0]*Comb_num
        for group in (largegroups + smallgroups):
            for i in range(Comb_num):
                pool[i] += group[i]
        print("Pool Size = %d" % sum(pool))
        scale = float(N) / float(sum(pool))
        print("Scale = %f" % scale)
        for i in range(Comb_num):
            pool[i] = ((pool[i] * scale))
        print("Pool Size after scale = %d" % sum(pool))
    PlotAll()
    print("OVER!")
    input()
    pass
```

‘‘‘

‘‘‘python

‘‘‘

B extension code

~~~
```python
import math
import random
import pylab
import time
outfile = "pool_extension.txt"
fig = pylab.figure()
K = 0.1 ## Global Death rate
R_small = 4.0 ## Resources for a small group
R_large = 50.0 ## Resources for a large group
R_medium= 27
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 = 6000 ## Population size
N_large = 40 ## Number of individuals in a large group
N_medium= 22
N_small = 4 ## Number of individuals in a small group
T = 150 ## Number of generations
t = 4 ## Number of time steps in groups
```
~~~

```
    <Figure size 432x288 with 0 Axes>
```

~~~
```python
Comb_num = 6
COOP_SM = 0
COOP_MD = 1
COOP_LG = 2
SELF_SM = 3
SELF_MD = 4
SELF_LG = 5
```
~~~

~~~
```python
fre_cs = list()
fre_cm = list()
fre_cl = list()
fre_ss = list()
fre_sm = list()
fre_sl = list()
large = list()
selfish = list()
small = list()
coop= list()
medium=list()
```
~~~

```
‘‘‘python
def Resource(group, R):
    addall = ( group[COOP_SM] * Gc * Cc ) +( group[COOP_MD] * Gc * Cc )+ ( group[COOP
    addall = R / addall
    resources = [0] * Comb_num
    resources[COOP_SM] = addall * group[COOP_SM] * Gc * Cc
    resources[COOP_MD] = addall * group[COOP_MD] * Gc * Cc
    resources[COOP_LG] = addall * group[COOP_LG] * Gc * Cc
    resources[SELF_SM] = addall * group[SELF_SM] * Gs * Cs
    resources[SELF_MD] = addall * group[SELF_MD] * Gs * Cs
    resources[SELF_LG] = addall * group[SELF_LG] * Gs * Cs
    return resources
‘‘‘


‘‘‘python
def Population(group, resource):
    group[COOP_SM] = (group[COOP_SM] + ( resource[COOP_SM] / Cc ) - K * group[COOP_SM
    group[COOP_MD] = (group[COOP_MD] + ( resource[COOP_MD] / Cc ) - K * group[COOP_MD
    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_MD] = (group[SELF_MD] + ( resource[SELF_MD] / Cs ) - K * group[SELF_MD
    group[SELF_LG] = (group[SELF_LG] + ( resource[SELF_LG] / Cs ) - K * group[SELF_LG
    return group
‘‘‘


‘‘‘python
def InitWrite():
    f = open(outfile, 'w')
    f.write("COOP_SM,COOP_MD,COOP_LG,SELF_SM,SELF_MD,SELF_LG\n")
    f.close()
    fre_cs = list()
    fre_cm = list()
    fre_cl = list()
    fre_ss = list()
    fre_sm = list()
    fre_sl = list()
    large = list()
    selfish = list()
    small = list()
    coop = list()
    medium=list()
‘‘‘


‘‘‘python
def WriteData(pool):
    f = open(outfile, 'a')
    f.write("%d,%d,%d,%d,%d,%d\n" % (pool[COOP_SM], pool[COOP_MD], pool[COOP_LG], poo
    f.close()
    fre_cs.append(pool[COOP_SM] / float(N))
    fre_cm.append(pool[COOP_MD] / float(N))
    fre_cl.append(pool[COOP_LG] / float(N))
    fre_ss.append(pool[SELF_SM] / float(N))
    fre_sm.append(pool[SELF_MD] / float(N))
    fre_sl.append(pool[SELF_LG] / float(N))
    large.append((pool[SELF_LG] + pool[COOP_LG] )/ float(N))
    selfish.append((pool[SELF_LG] +pool[SELF_MD]+ pool[SELF_SM] )/ float(N))
    small.append((pool[SELF_SM] + pool[COOP_SM] )/ float(N))
    medium.append((pool[SELF_MD] + pool[COOP_MD] )/ float(N))
    coop.append((pool[COOP_LG] + pool[COOP_MD]+pool[COOP_SM] )/ float(N))
    pass

‘‘‘


‘‘‘python
def PlotAll():
    pylab.figure(fig.number)
    pylab.xlabel(”Generation”)
    pylab.ylabel(”Global frequency”)
    x=range(T)
    pylab.plot(x, large, ’k:’, label=”Large Group Size”)
    pylab.plot(x, medium, ’g:’, label=”Medium Group Size”)
    pylab.plot(x, small, ’r:’, label=”Small Group Size”)
    pylab.plot(x, selfish, ’k-’, label=”Selfish resource usage”)
#     pylab.plot(x, coop, ’r-’, label=”Cooperative resource usage”)
    pylab.legend(loc=9, bbox_to_anchor=(0.5, -0.15))
    pylab.show()
    pylab.draw()
    pylab.xlabel(”Generation”)
    pylab.ylabel(”Global genotype frequency”)
    pylab.plot(x, fre_cs, ’g-’, label=”Cooperative+Small”)
    pylab.plot(x, fre_cm, ’b-’, label=”Cooperative+Medium”)
    pylab.plot(x, fre_cl, ’r-’, label=”Cooperative+Large”)
    pylab.plot(x, fre_ss, ’g:’, label=”Selfish+Small”)
    pylab.plot(x, fre_sm, ’b:’, label=”Selfish+Medium”)
    pylab.plot(x, fre_sl, ’r:’, label=”Selfish+Large”)
    pylab.legend(loc=9, bbox_to_anchor=(0.5, -0.15))
    pylab.show()
    pylab.draw()
    pass
‘‘‘


‘‘‘python
def Test():
    test = [6.0,6.0,6.0,6.0,6.0,6.0]
    r = Resource(test, R_large)
    print (”Group :”)
    print (test)
    print(”Resources: ”)
    print (r)
    Population(test, r)
    print (”Group :”)
    print (test)
#     input()
    pass
‘‘‘


‘‘‘python
Test()
‘‘‘
```

```
    Group :
    [6.0, 6.0, 6.0, 6.0, 6.0, 6.0]
    Resources:
    [5.172413793103448, 5.172413793103448, 5.172413793103448, 11.49425287356322, 11.49
    Group :
    [57.12413793103448, 57.12413793103448, 57.12413793103448, 62.871264367816096, 62.8
```

~~~
```python
if "__main__" == __name__:
    InitWrite()
    pool = list()
    for i in range(Comb_num):
        pool.append( float(N / Comb_num ) )
    print (pool)
    for g in range(T):
        print("GENERATION %d" % g)
        WriteData(pool)
        smallgroups = list()
        mediumgroups = list()
        largegroups = list()
        sm = int((pool[COOP_SM] + pool[SELF_SM]) / N_small)
        md = int((pool[COOP_SM] + pool[SELF_SM]) / N_medium)
        lg = int((pool[COOP_LG] + pool[SELF_LG]) / N_large)
        if sm:
            p_sm_coop = pool[COOP_SM] / ( pool[COOP_SM] + pool[SELF_SM])
            for i in range(sm):
                group = [0.0] * Comb_num
                for i in range(N_small):
                    if (random.random() < p_sm_coop):
                        group[COOP_SM] += 1
                    else:
                        group[SELF_SM] += 1
                smallgroups.append(group)
        if md:
            p_md_coop = pool[COOP_MD] / ( pool[COOP_MD] + pool[SELF_MD])
            for i in range(md):
                group = [0.0] * Comb_num
                for i in range(N_medium):
                    if (random.random() < p_md_coop):
                        group[COOP_MD] += 1
                    else:
                        group[SELF_MD] += 1
                mediumgroups.append(group)
        if lg:
            p_lg_coop = pool[COOP_LG] / ( pool[COOP_LG] + pool[SELF_LG])
            for i in range(lg):
                group = [0.0] * Comb_num
                for i in range(N_large):
                    if (random.random() < p_lg_coop):
                        group[COOP_LG] += 1
                    else:
                        group[SELF_LG] += 1
                largegroups.append(group)

        for group in largegroups:
~~~

```
            for _t in range(t):
                rl = Resource(group, R_large)
                Population(group, rl)


        for group in mediumgroups:
            for _t in range(t):
                rm = Resource(group, R_medium)
                Population(group, rm)
        for group in smallgroups:
            for _t in range(t):
                rs = Resource(group, R_small)
                Population(group, rs)#last value is preserved
        pool = [0.0]*Comb_num
        for group in (largegroups + mediumgroups +smallgroups):
            for i in range(Comb_num):
                pool[i] += group[i]
        print("Pool Size = %d" % sum(pool))
        scale = float(N) / float(sum(pool))
        print("Scale = %f" % scale)
        for i in range(Comb_num):
            pool[i] = ((pool[i] * scale))
        print("Pool Size after scale = %d" % sum(pool))
    PlotAll()
    print("OVER!")
#     input()
    pass
```

```python
```