The Importance of Group-Size Preferences for the Evolution of Cooperation Under the Conditions of Individual Selection

COMP6202 Assignment II

Paper: *Powers, S. T., Penn, A. S. and Watson, R. A. (2007) Individual Selection for Cooperative Group Formation*

Abstract

Bacteria and infectious diseases pose a significant threat to global health. Though research into like organisms, such as bacterial bio-film, has continued growing, our understanding of them is become evermore-so complex. Models impose benefit in this regard. Their ability to simulate particular environments is a powerful tool in gasping the significance of certain traits in differing species. Within this paper we reproduce a model which imposes individual selection, where the consequent environmental conditions are defaulted to impede the evolution of cooperation. Despite this, we find that cooperation can prevail. We then furthered this model by evaluating its (strict) aggregation and dispersal process, arguing that organisms are unlikely to be exact in their preferences of group size, thus we provided individuals with the chance of joining a discrete range of group sizes; the size of the group being dependant on their individual preferences. Results show that cooperation can evolve and is inevitably more controlled than selfish behaviour. We also found that individuals with a general preference for small groups are the only redeeming factor in determining whether cooperation will prevail. Considering we can only design successful cooperative strategies when small-groups are selected for, it would be interesting to see a model where large-cooperative genotypes triumph.

1 Introduction

The paper we followed in this assignment, **?**, outlines several misleading assumptions in prior work which, then, limited the explanatory power of models (and adjoint parameter settings) focused on driving cooperative strategy within spatially structured (haploid) populations. Previously, the model-environments imposed the conditions which regulate selection (to favour cooperation over selfish behaviour); rather, the proposed model lends the power of selection to the individuals to provide reason for group formation[1]. Thus, answering the question, which conditions and behaviours will evolve when individuals have the power to modify their own environment? What is interesting is the perspective which *niche construction* (NC) provides to the interpretation of real organism aggregation and dispersal processes. However, from NC is derived *social niche construction* (SNC), which goes a step further to "bridge the gap between individual and group selectionists", **?**. SNC suggests that individual adaptation strengthens group cohesion, thus reinforces group-pressure for individual adaption. Hence, SNC provides context and reason for when group-level selection pressures dominate; proposing that groups should not be simply viewed in context of the environment. Though **?** was published before the extension of SNC, it still maintains aspects of SNC. Within the model, individuals are given the power of selection by declaring a preference for group-size (small/large phenotype) and when individual adaption occurs (thriving selfish/cooperative phenotypes) individual group-size preference become more influential in selection, thus group-level pressure persists.

To provide more technical context, the model specifies a genotype by two traits: (1) [Cooperative/Selfish] resource usage, (2) Preference for the group size which the individual will join [Small/Large]. The former trait imposes the conditions necessary to stimulate selfish and cooperative action, hence cooperative individuals regulate their resource intake (reducing waste) to benefit the group. On the other hand, the latter trait lends the individual the choice of environment, whereby there exists the intrinsic advantage of a large group's resource influx being much greater than a small group's influx.

From a migrant pool containing all individuals, small and large groups are formed by assigning individuals to random groups of the individuals preferred size (if the individuals that remain can't form a group they die out). Within each group a reproduction cycle lasting T epochs is simulated, whereby individuals asexually reproduce clones at each time-step, t. Equation 1 illustrates how the number of individuals, $n_i(t)$, pertaining to class $i\epsilon[Selfish/Cooperative]$ within a group evolves at a time-step $t+1$, where r_i represents the resource influx for those individuals, C_i represents the consumption rate of the individuals and K represents a fixed death-rate. A small correction has been made to Equation 2 in the paper, whereby the resource influx of the individuals represented by n_i follows the growth of the group through each epoch of reproduction, thus r_i has dependency on time[2]. The reinterpreted equation to determine class-specific resource influx is denoted in Equation 2.

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

$$r_i(t) = \frac{n_i(t)C_iG_i}{\sum_j n_j(t)C_jG_j}R \tag{2}$$

In addition to reproduction, the paper follows *Wilson's trait-group aggregation and dispersal model.* This

[1]This is known as *niche construction*, **?**

[2]Whether this was the initial intention or not, we thought best to clarify.

method offers a way of emulating the dispersal and regrouping process of individuals, for example, in bacterial biofilm formation, ?. ? retains the purpose of simulating realistic features of existing organisms to benefit our understanding of them, particularly Equation 2 which is motivated by bacteria colony growth.

The significance of such research is unassumingly important. "(80%) of infectious and persistent bacteria are biofilm-formers, ?, and that in nature microorganisms are actually forming biofilms, ?", ?. ? provides further discussion as to particular biofilm research and application, outlining there is a lack of research towards biofilm formations, comparative to platonic bacteria. Considering the impact research in this field has on global health, it is becoming more imperative to solve related problems as human population continues to rise ?, thus is at risk ?.

2 Model Implementation

In this section we elaborate on particular aspects of model implementation and parameter settings, to later discuss the results of the competitive environment when all four traits are initially equally available. The parameter settings are detailed in Table 1.

The model is structured as an iterative process which follows the following sequence (modified from ?).

1. **Initialisation:** Initialise the global migrant pool with N individuals
2. **Group Aggregation:** Assign individuals in the migrant pool to randomly selected groups of their preferred size.
3. **Reproduction:** Perform T epochs of reproduction (cloning of successful individuals) as per Equation 1.
4. **Dispersal:** Return the resulting sub-populations to the migrant pool, i.e. reform the global migrant pool.
5. **Rescale to Maintain Carrying Capacity:** To realistically simulate an enclosed environment, we need to rescale the resulting population to the environment's capacity, N
6. Iterate from step 2 for τ generations.

To stimulate the competitive environment, selfish phenotypes are provided the chance of receiving a larger proportion of the available resource (as a result of higher growth rate $G_s > G_c$). Further, to simulate selfish behaviour, individuals who carry this trait consume more than cooperators. Considering only this, one may jump to the conclusion that selfish individuals have an advantage thus should be more dominant, however as we found out in ? selection pressures surprisingly favour both small and cooperative individuals.

We could hypothesise that the cooperative allele plays a larger role in reproduction than we originally thought. By inspecting the "$\frac{r_i(t)}{C_i}$" term from Equation 1[3], we can consider the extreme cases for each genotype: (1) when a small group is comprised of solely selfish individuals, the number of clones as a result of resource influx is 200 (in the first epoch of the reproduction cycle), (2) when a small group is comprised of cooperators, the number of clones as a result of resource influx is 222 (marginal increase of 11%), (3) for a large and completely selfish group, this return is 2500, (4) for a large and completely cooperative group, this return is 2,777 (another marginal increase of 11%). Evidently one could be led to believe that cooperative phenotypes are more beneficial for population prosperity as they are not only more efficient consumers, they are also better reproducers (11% better with respect to group size). However if one was to take this stance, one would also have to assume the position that large-cooperative genotypes are overall the best reproducer, thus should be favoured by selection. Alas, this is not telling of the result.

3 Results

In this section we aim to reproduce the results from Section 3.2 ?, hence a reproduction of Figure 1.

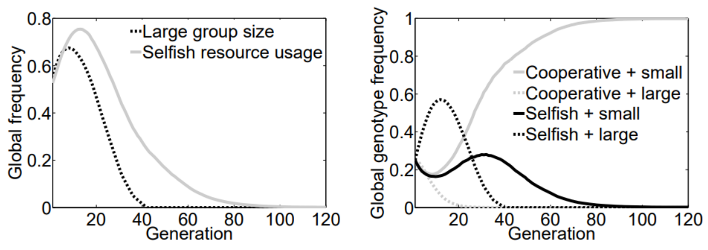

Figure 1: Left hand: The average environment and strategy through time. Right hand: change in genotype frequency over τ generations, (Figure 2 ?)

The results of our reproduction are illustrated in Figure 2. As expected, the results are identical to those shown in ?. We demonstrates how selective-pressures react to a hike in large-selfish genotypes by selecting for small-group phenotypes, before completely favouring the small-cooperative genotype later in the process. The reasoning behind this is better understood under further inspection of the right-hand plot. Selfish cheaters don't reach fixation (hence, environmental equilibrium) as they thrive in diverse groups, so when they begin to dominate groups (specifically when large cooperators die out), they succumb to selection pressures which favour smaller populations. Thereafter, small cooperators become more advantageous and are more intensely exploited.

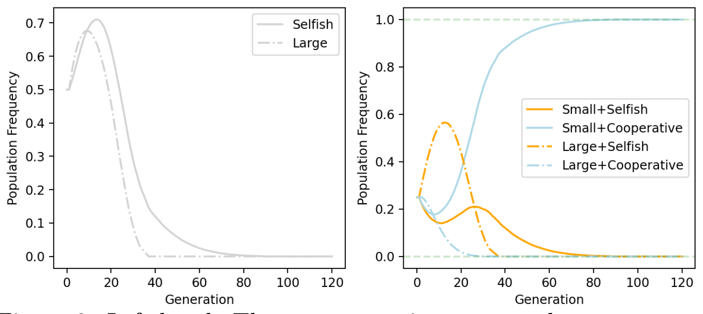

Figure 2: Left hand: The average environment and strategy through time. Right hand: change in genotype frequency over τ generations

[3]This term is largest contributor to clone reproduction within this expression, hence our decision to investigate.

Parameters	Value
Consumption rate (cooperative), C_c	0.018
Consumption rate (selfish), C_s	0.02
Growth rate (cooperative), G_c	0.1
Growth rate (selfish), G_s	0.2
Death rate	0.1
Num. of Epochs for reproduction, T	4
Global Population Size, N	4000
Num. of Generations, τ	120
Group size (small), N_s	4
Group Size (large), N_l	40
Resource Influx (small), R_s	4
Resource Influx (large), R_l	50

Table 1: Parameter settings for model implementation

What we can say as a result of this is that selfish phenotypes thrive in large groups, yet to their own demise. More abstractly, selfish phenotypes in relation to the state of the environment are conclusively deemed unfavourable. Hence, when no external environmental parameters are controlled to influence group selection, individual-choice becomes the basis for selection and cooperation is favoured in the long-term.

What we can not say is that this is the case in a realistic, natural, environment comprised of a variation of cheaters and cooperators. To a degree, the model we replicated represents the generalised case for this type of simulated scenario. So what would happen if conditions weren't so fixed?

4 Extension Work

4.1 Discussion

In this section we propose avenues for further investigation, as a result of concluding analysis on the replicated model.

Considering the publication date of **?**, we can deduce that research into the types of organisms the paper is trying to simulate was sparse. For example, the number of papers published on the topic of biofilms prior to the release of this paper is significantly smaller than the number of papers following the publication, **?** (Figure 3). Therefore it would be unreasonable to assume that a more realistic interpretation of the naturally occurring environments was available.

By investigating the model parameters, we can tell that several aspects of the natural evolution are amiss. The first is the missing element of random mutation, which should persist in some manner. With a specific-bacteria in mind, say *Pseudomonas aeruginosa*, an appropriate mutation rate with respect to a global population would be rounghly 6%, **? ?**. Hence, one could investigate the effects of varying mutation rate with respect to actual organisms, to discern perhaps the volatility of cheaters in differing environments. Similarly, this could be done with the death rate parameter, K, **?**.

The second aspect which is left unexplained is the lack of variation which is assumed by this model. Specifically we criticise the construction of group-size preference trait, which could be prohibiting variation from playing its role in selection. Considering the aims of the original paper were centered on bettering the simulation of like-organisms by improving selection, the construction of this selection criteria is simplistic, an opinion shared by the same authors thereafter, **?**.

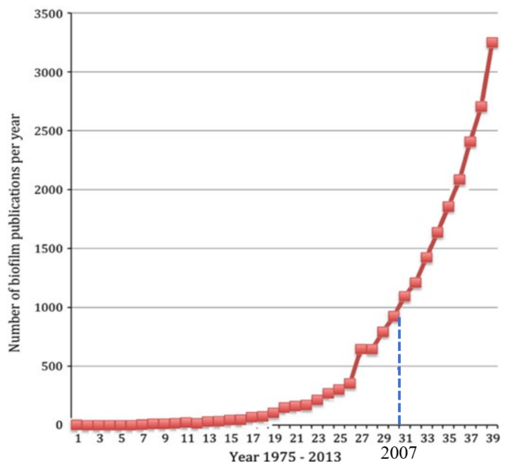

Figure 3: The number of papers published each year on biofilms from 1975-2013, **?** (2007 is the publication year of **?**)

We have discussed in Section 1 how realistic interpretation of organisms is perhaps in the best interest of research for global-health. Consequently, one could extend the paper by implementing and organism-specific variation of the proposed model model. For example, one could investigate varying the type of aggregation and distribution that follows the *P. aeruginosa, Klebsiella pneumoniae* and *Flavobacterium* bacterium **?**. As opposed to Wilson's aggregation and dispersal process, individuals are free to move between groups during reproduction, thus group sizes are allowed to vary (to the limit of the local environment). The caveat with this particular implementation is that as well as natural death, individuals can be cast out/detached from the network (eventually they dissolve into the environment similarly to how individuals which aren't grouped during the re-grouping stage die out, **?**).

4.2 Model Implementation

To extend the model we have chosen to focus on simulating and testing new interpretations of the trait for group-size preference. Prior work (by the same authors as **?**) has already been done on extending this property, **? ?**. Their motivation being that (then) existing models tended to simplify the structure of the trait for group size preference to a bi-allelic locus. So, the trait was redefined as a multi-allelic integer locus where size preference is still fixed. This suggests a higher variation in the initial population size, yet the results explain that small-cooperative variants are still selected for. What is interesting is that when a population relies on mutation to discover group-sizes (where the initial population has the same group size allele), the tendency for the final (expected) cooperative population is a group size of one, **?**. As reasoned by the authors, once cheaters are purged there is no competition driving the need for groups with the cooperative trait, thus fitness tends to single person groups.

We would like to take the idea of a multi-allelic integer locus for group size preference a step further. We hypothesise that an abstract trait such as size preference does not naturally come in fixed value form, hence individuals (particularly single-cell organisms which form micro-colonies, such as bacterial biofilm **?**) do not have complete understand of what exact group sizing is. Thus, if a preference as simple as *large* or *small* does exist, it will come with some (hopefully quantifiable) margin of error. Within our model we have structured this trait as a distribution of size-preferences with a tight deviation, $\sigma_{(\mathbf{size})} < [1,3]$ around a given mean size preference, $\mu_{\mathbf{size}}\epsilon[4-6, 35-45]$ for any vector $\mathbf{x}_{(\mathbf{size})} = [x_{(small)},\ \ x_{(large)}]$. This will encompass both a margin of error for what an individual may classify as 'small' or 'large' and a margin of error for their estimation of this size. Hence, an individual will select and form a group who's size is bounded by all its member's, size-preference distribution (technical detail to follow).

Within our extension we would also like to minimise the tendency for single group cooperators as we reason this defeats the purpose of cooperative behaviour[4], thus we bound the minimum group size to 2.

Thus, each individual will select for three traits:

1. **Abstract Group Size Preference:** $(size)\ \ \epsilon\ \ [Small/Large]$

2. **Preference (Normal) Distribution:** $N(\mu_{(size)}, \sigma_{(size)})$

3. **Resource usage:** $[Cooperative/Selfish]$

The abstract group size preference trait will indicate which general group size the individual selects for. Further, the preference distribution will be selected for, within the bounds determined by the abstract group size preference (specifically bounded by, $8 > \mu_{(small)} + \sigma_{(small)}$, $\mu_{(small)} - \sigma_{(small)} > 1$, $49 > \mu_{(large)} + \sigma_{(large)}$ and $\mu_{(large)} - \sigma_{(large)} > 31$). This means we code for 22 possible genotypes in our population, 16 of which are part of the large group (group sizes can vary from 2 to 7 and 32 to 48). Our logical assumption here is that individuals can't directly select for specific group size (even under assumptions of multi-allelic integer locus) and when faced with a larger group size individuals have a harder time estimating size. Consequently, the group-size margin of error should have some proportionality with abstract group size; even intellectual beings such as ourselves find it more difficult to estimate the number of objects in an environment, the more objects are added. So by reason, there should be a larger distribution of "large" group size preferences. The dynamics of this model resembles the method of aggregation in bacterial biofilm, **?**[5]

We maintain the same testing hypotheses as in our previous model; we want to understand what is chosen for, selfish or cooperative behaviour, when the individuals in a population have extensive control over their selective environment (subsequently, what is chosen for if an individual's interpretation of group-sizing varies). As seen in **?**, we hold the expectation that the abstract group size preference will select for the smallest size and distribution, however it will be interesting to see if this expectation hold up for scenarios such as individuals having a skewed interpretation of size.

The structure of this new algorithm resembles the prior algorithm with key differences:

1. **Initialisation:** Initialise the global migrant pool with N individuals, (equal distribution of selfish, cooperative, small and large phenotypes).

2. **Group Aggregation:** Assign individuals to randomly selected groups. A group's size must agree with all member's preference distribution and a group is done forming when the maximum number of members is reached as a result of the distributions of the existing group members. (i.e. when the group size reaches the largest possible size preference of the member with the lowest mean size preference.)

3. **Reproduction:** Perform T epochs of reproduction as per Equation 1. If selfish/cooperative phenotypes experience increasse by s, we randomly select s individuals (of that phenotype) to clone. Otherwise if a phenotype experiences a decrease by s, we randomly select s individuals to expire.

4. **Dispersal:** Return the resulting sub-populations to reform the global migrant pool.

5. **Rescale to Maintain Carrying Capacity:** To realistically simulate an enclosed environment, we need to rescale the resulting population to the environment's capacity, N.

6. Iterate from step 2 for τ generations.

Lastly, we need to consider the distribution of resources across differing group sizes. Equation 3 expresses the resource influx of a group sized, n, dependant on the resource intake per capita, r. As before, group sizes of 4 and 40 have resource intakes of ~ 4 and ~ 50, respectively. The reasoning behind this equation is laid out in Appendix A.

$$R(n) = n \cdot r_0 \cdot (1.05)^{log_2(n)} \tag{3}$$

[4]Perhaps this is simply the issue with emulating cooperative behaviours on a machine. Perhaps this is an attribute left for extension in another paper.

[5]Note that we have not catered for misnomer groups which appear have size 8-28 and more than 48. This was to reduce bias in determining a strict boundary for small/large groups, however these regions are discussed.

4.3 Results

In this section we discuss results we found while varying the parameters we set to explore. *To ensure reproducibility, in this section we use a random seed of* 7.

4.4 A Continuation of Reproduction Figure 2

Though the extension to the original model redefines a rather important aspect of the algorithm, the essence still maintains. The model, if placed under the same conditions as described in ?, should produce the same results as in Figure 1.

Consequently, we bounded our model by ensuring all individuals held mean preference values $\mu_{size} = [4, 40]$ and all groups held the same fixed quantities. Figure 4 illustrates the results. As expected, the small-cooperative genotypes become dominant in the same manner as the prior model. Despite this the figure shows that under the new conditions, large-selfish genotypes have a shorter "rise-to-fame". As a result, the small-cooperative individuals take less generations to out-compete all other genotypes; most likely a result of higher selective pressures.

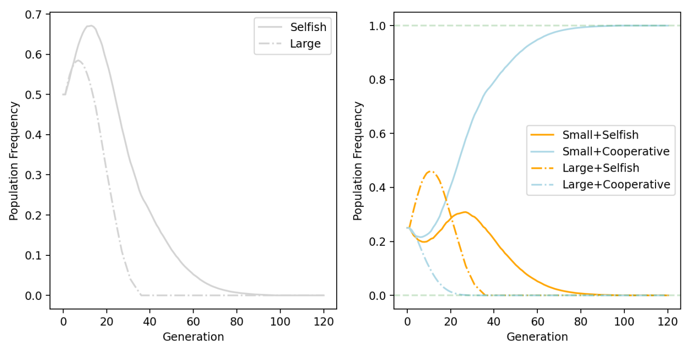

Figure 4: Reproduction o Figure 1

Provided this was a replication test, we could pin the change in frequency of selection down to the difference in the resource influx parameter, $R(n)$ from the prior models given parameters ($[R_{small} \quad R_{large}] = [4 \quad 50]$). Nonetheless, there is significance in a replication of environmental behaviour.

4.5 Preliminary Experimentation

Under the reasonable conditions for group-sizing and resource distribution laid out in Section 4, we hold the expectation that the selfish phenotype will prevail. As per understanding why this may be the case (as opposed to selecting for co-opertative individuals), we only need to look as far as Figure 1 ? to reason that any group size ≥ 6 will not select for cooperative individuals. Considering there is also an advantage to large group sizes, small size-preference phenotypes may tend the higher end of the defined size range or small groups.

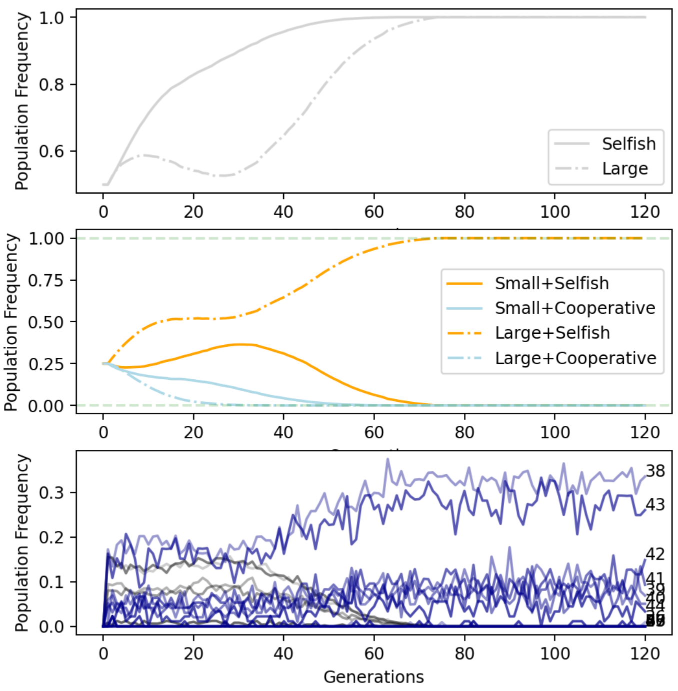

Figure 5: Top: Frequency of advantageous alleles. Middle: Frequency of generalised genotypes. Bottom: Population frequency measured over group-size (Grey: Small-trait, Blue: Large-trait)

Figure 5 provides evidence for our hypothesis, yet we would argue the most interesting result of this simulation is the difference in group-sizes which are finally selected for, (38 and 43). Though a great variation of large phenotypes still exists, group-sizing tends to the smaller and larger end of the available group-sizes, suggesting the large phenotypical population divides. Comparative to our analysis in the prior subsection, we can see how essential interpreting group-sizing is to the success of cooperative individuals. If organisms show low-spatial awareness, thus become less tolerant to group sizing then selection will favour large-selfish genotypes; which suggests that size-interpretation traits must evolve with behaviour for selection to favour cooperation.

4.6 Ensuring Cooperation

As with the section on preliminary experimentation in ?, we want to understand which reasonable parameter set exists to ensure cooperation is selected for. To accomplish this we decided on iterating our algorithm over variants of the original gene as described in Section 4.2. We defined four sets of genes where small/large phenotypes have a preference distribution, $[\mu_{size}, \sigma_{(size)}(= [3, 1])]$ bounded by: (1) $\mu_{size} = [4-5, 37-43]$, (2) $\mu_{size} = [2-5, 37-43]$, (3) $\mu_{size} = [4-5, 30-50]$, (4) $\mu_{size} = [2-5, 30-50]$. This provided the opportunity to investigate both isolated and shared cases where an individual's interpretation of their group-size preference is varied. Hence, we are evaluating scenarios where organisms may find it easier or harder to interpret a group-size preference characteristic. It is worth noting that in consequence the discussion in Subsection 4.5, small group sizes are bounded by < 6.

The result of these tests are illustrated in Figure 6. Interestingly, when a small group is more varied, (curves (2) and (4)), selective control favours small cooperative individuals in similar (strict) manners. When this is not the case selective frequency tends to small-selfish genotypes, as seen by the frequency of large individuals in scenario (3) going extinct while the frequency of selfish individuals continues to rise (though hesitantly). It is only when both small and large groups are tightly varied that both selfish and large phenotypes more frequently selected.

To make sense of this, again we can look to Figure 1 **?** for similarities. It is clear when small-groups are "smaller" (within the realms of the original model) that they have a higher change of being selected for, hence if we introduce a potential range of small group sizes (as we have with the extended model), there is reason to assume that the potential for selection (favouring cooperation) is increased. This assumption holds so long as the small groups are reasonably "small" (size < 6). By this logic, inner-group variance should be large, however Figure 7 shows that individuals finally select for groups at the bounds of the existing range of group sizes, not necessarily cohering to what is imposed within the algorithm. That is to say large groups not necessarily selected for. We saw similar behaviour in Figure 5.

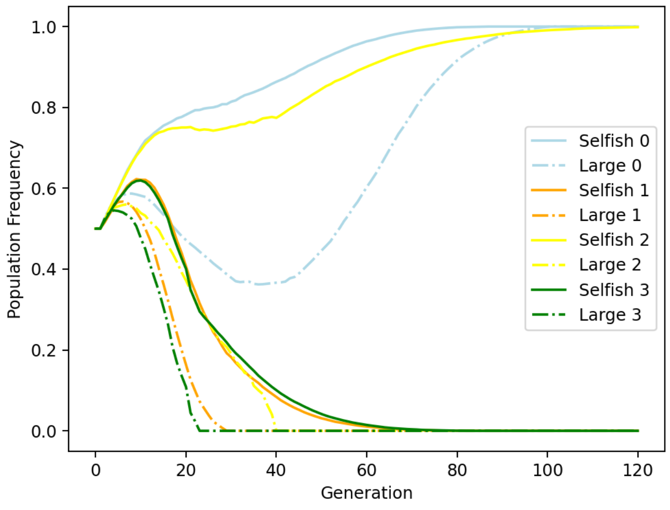

Figure 6: Frequency of naturally advantageous alleles. (Blue: Results from genotype variant (1), Orange: Genotype variant (2), Yellow: Genotype variant (3), Green: Genotype variant (4))

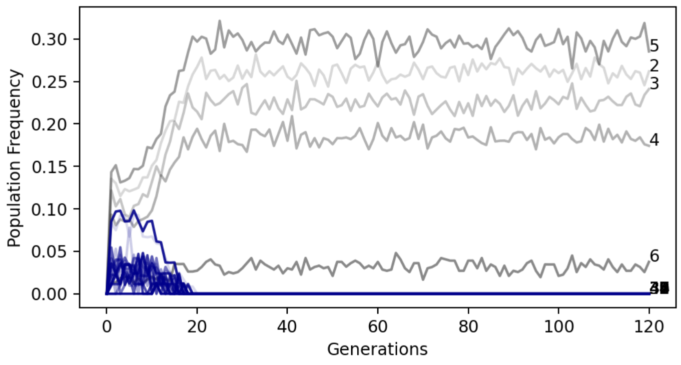

Figure 7: Frequencies of group-sizes selected under the conditions of Genotype variant (1)

4.7 Conclusion

Within the natural sciences we have come far enough to understand that selfish/un-selfish behaviour can be considered a genetic trait, (**?**, **?**). Whether it truly the case or not, it is still a behavioural trait which can, like genes, be passed on through methods other than sex, such as influence and experience. Hence, the utility of understating the evolution of traits such as selfishness and altruism can be beneficial in comprehending how organisms interact[6]. As we discussed in Section 1, the modelling behind such traits bears great benefit to global healthcare. Thus, the investigation into better defined, perhaps more flexible, models is more beneficial.

This is the premise of **?**, which we have readily evaluated in previous sections. As discussed, we went a step further in the same direction, evolving the selection criteria to vary inner group-size preference. Hence, we provided individuals with the extended choice of joining groups which can only grow to their size preferences. This inevitably creates divide within the large/small groups which, as we saw in Figure 5 and Figure 7, selects for individuals (and subsequently for groups) with size preferences at each end of the small and large group distributions.

In addition, we determined that successful selection of the cooperative trait is independent of the variation which may exist in large groups; hence it is dependant on how individuals in smaller groups interpret and select for smaller group sizes. This is significant as it suggests that populations which have a growing tendency to cooperate (on a global scale) as a result of smaller micro-colonies also tend to behaviour which is less seeming of cooperation, i.e. individuals are inevitably less willing to cooperate en-mass. It would be intriguing to see a model, under the conditions of SNC, which nonetheless maintains the essence of *cooperation en-mass.*

Conclusively to this extension, there are other particular aspects which would be interesting to further investigate. For example, how would mutation play a role in this new environment? A relatively simple extension, though perhaps necessary if any further work was to be done on this. Like with **?**, it would also be interesting to see how groups emerge if the size preference distributions overlapped. Would individuals select for larger or smaller groups despite the group they initially preferred? Perhaps even further, is there a method we can construct within our environment to coerce selfish individuals to act altruistically?

[6]We can go as far to say it can help us understand how such organisms perceive their environment.

A Derivations

A.1 Deriving the Resource Influx dependant on group size

Consider the distribution of resources as per the general statements, “a group that is twice as large receives an extra 5% per capita resource influx”, **?**. This would suggest that,

$$r_{2s} = r_s \cdot 1.05$$

where s represents the size of the smaller group. And further,

$$r_{4s} = r_{2s} \cdot 1.05 = r_s \cdot (1.05)^2$$
$$r_{8s} = r_{4s} \cdot 1.05 = r_s \cdot (1.05)^3$$

Hence, if we set s = 1, then the expressions can be reinterpreted as,

$$r_2 = r_1 \cdot (1.05)^1$$
$$r_4 = r_1 \cdot (1.05)^2$$
$$r_8 = r_1 \cdot (1.05)^3$$

which leads to the following equation,

$$r_s = r_1 \cdot (1.95)^m$$

Where m is defined as,

$$m = log_2(s) \Rightarrow r_s = r_1 \cdot (1.95)^{log_2(s)}$$

As $R(s) = r_s \cdot s$, we can define Equation 3:

$$R(n) = n \cdot r_0 \cdot (1.05)^{log_2(n)}$$

B Source Code (Original Model)

B.1 main_noext.py

```
from environment_noext import Environment
from gene_noext import Gene

# define popsize, generations and fixed group sizes
global POPSIZE
POPSIZE = 4000
global small_size
global large_size
small_size, large_size = 4, 40
# t = 4 (reproduce 4x in group before migrant reformation)
epochs = 4

def Initialise(popsize):
    """Initialise population of selfish and cooperative individuals, and small and large preferences
    """
    pop = []
    for i in range(popsize):
        # Split pop size into 4 genotypes
        if i % 8 in [0,1]:
            # isSmall and iscoop
            ind = Gene(True, True)
        elif i % 8 in [2,3]:
            # SC
            ind = Gene(True, False)
        elif i % 8 in [4,5]:
            # LC
            ind = Gene(False, True)
        elif i % 8 in [6,7]:
            # LS
            ind = Gene(False, False)
        pop.append(ind)
    return pop

if __name__ == '__main__':
    ratios = []

# 1 Initialisation
    pop = Initialise(POPSIZE)
    env = Environment(pop, POPSIZE)
    # Set first set of ratios (frequencies) for plotting
    env.save_ratios()

    for i in range(120):
        print('Iteration ', i)

    # 2 Group Formation
        env.group_formation()

    # 3 Reproduction
        env.reproduce()

    # 4 Migrant Pool Formation
        env.migrant_pool_formation()

    # 5 Maintain Global Capacity
        psize = env.get_popsize()
        print(f'Final Pop Size {psize}')
        if psize > POPSIZE:
            env.rescale()
    # Plot results
    env.plotting()
```

B.2 environment_noext.py

```
from gene_noext import Gene
import group_noext as grp
import random
import matplotlib.pyplot as plt

def filter_pop_by_size(pop: list = []):
    popS, popL = [], []
    for p in pop:
        if p.issmall:
            popS.append(p)
        elif not p.issmall:
            popL.append(p)
    return popS, popL

class Environment:
    small_size = 4
    large_size = 40
    t = 4

    def __init__(self, pop, N):
        self.N = N
        self.Groups = []
        self.pool = pop
        self.poolSize = len(pop)
        self.hist_ratios = []

    def get_popsize(self):
        return len(self.pool)

    def group_formation(self):
        """Form small and large groups by randomly shuffling the subset population (small and large gene s
        extracting final index. Groups are sized by ‘self.large_size‘ and ‘self.small_size‘

        Outcome
        -------
        Generate a list of all group objects ‘self.Groups‘
        """
        self.Groups = []
        popS, popL = filter_pop_by_size(self.pool)
        print(f’Population Size: {len(self.pool)}’)
        Rs, Rl = 4, 50
        num_small = int(len(popS) / self.small_size)
        num_large = int(len(popL) / self.large_size)

        # Group small individuals
        for i in range(num_small):
            tpop = []
            for j in range(self.small_size):
                random.shuffle(popS)
                tpop.append(popS[-1])
                popS.pop(-1)
            self.Groups.append(grp.Group(tpop, True, R=Rs))
        # Group Large individuals
        for i in range(num_large):
            tpop = []
            for j in range(self.large_size):
                random.shuffle(popL)
                tpop.append(popL[-1])
                popL.pop(-1)
            self.Groups.append(grp.Group(tpop, False, R=Rl))
        del popS, popL

    def reproduce(self):
        """Run each Group object’s reproduction through ‘t‘ time steps
        """
        Groups_ = []
        for g in self.Groups:
            for i in range(self.t):
                g.get_ri()
                g.reproduce()
            g.regenerate_pop()
            Groups_.append(g)
        self.Groups = Groups_
        self.save_ratios()

    def migrant_pool_formation(self):
        """Return all individuals to the migrant pool, ‘self.pool‘
        """
        pool = []
        popSize = 0
        for g in self.Groups:
            for p in g.pop:
                pool.append(p)
                popSize += 1
        self.pool = pool

    def rescale(self):
        """Determine the genotype population split (ratio, ‘fi‘ for i in ss,sc, ls, lc).
        """
        # Init. number of genotypes index: (0) ss, (1) sc, (2) ls, (3) lc
        ni = [0. for i in range(4)]
        self.poolSize = len(self.pool)
        for p in self.pool:
            i = 0
            if p.iscoop:
                i += 1  # index (1) or (3)
            if not p.issmall:
                i += 2  # index (0) or (2)
            ni[i] += 1.
        # Determine the ratio of each genotype
        fi = [ni[i] / self.poolSize for i in range(4)]
        ni_ = [int(fi[i] * self.N) for i in range(4)]
        pool = []
        for i in range(4):
            for j in range(ni_[i]):
                # issmal, iscoop
                if i == 0:
                    pool.append(Gene(True, False))
                elif i == 1:
                    pool.append(Gene(True, True))
                elif i == 2:
                    pool.append(Gene(False, False))
                elif i == 3:
                    pool.append(Gene(False, True))
        self.pool = pool

    def save_ratios(self):
        # Init. number of genotypes index: (0) ss, (1) sc, (2) ls, (3) lc
        ni = [0 for i in range(4)]
        self.poolSize = len(self.pool)
        for p in self.pool:
            i = 0
            if p.iscoop:
                i += 1  # index (1) or (3)
            if not p.issmall:
                i += 2  # index (0) or (2)
            ni[i] += 1
        fi = [ni[i] / self.poolSize for i in range(4)]
        self.hist_ratios.append(fi)
        print(f'Pop Ratios: \n    SS {fi[0]}\n    SC {fi[1]}\n    LS {fi[2]}\n    LC {fi[3]}')

    def plotting(self, fignum:int=0):
        f, sub = plt.subplots(1,2)
        ss, sc, ls, lc = [], [], [], []
        selfish, large = [], []

        for r in self.hist_ratios:
            ss.append(r[0])
            sc.append(r[1])
            ls.append(r[2])
            lc.append(r[3])
            selfish.append(r[0]+r[2])
            large.append(r[2]+r[3])

        sub[1].plot(ss, label='Small+Selfish', color='orange')
        sub[1].plot(sc, label='Small+Cooperative', color='lightblue')
        sub[1].plot(ls, label='Large+Selfish', color='orange', linestyle='-.')
        sub[1].plot(lc, label='Large+Cooperative', color='lightblue', linestyle='-.')
        sub[1].axhline(1, color='green', alpha=0.2, linestyle='--')
        sub[1].axhline(0, color='green', alpha=0.2, linestyle='--')
        sub[1].set_xlabel('Generation')
        sub[1].set_ylabel('Population Frequency')
        sub[1].legend()

        sub[0].plot(selfish, label='Selfish', color='lightgrey')
        sub[0].plot(large, label='Large', color='lightgrey', linestyle='-.')
        sub[0].set_xlabel('Generation')
        sub[0].set_ylabel('Population Frequency')
        sub[0].legend()

        plt.show()
```

B.3 group_noext.py

```
from gene_noext import Gene

class Group:
    # Growth & Consumption Rates (and product)
    Gs, Gc = 0.02, 0.018
    Cs, Cc = 0.2, 0.1
    GsCs, GcCc = Gs * Cs, Gc * Cc
    # Death rate
    K = 0.1

    def __init__(self, pop: list = [], size: bool = False, R: float = 0.0):
        """Initialise group of individuals

        Attributes
        ----------
            pop : list of Gene, list of individuals
            size : bool, True for small, False for large
            R : float, resource influ for group
        """
        self.pop = pop
        self.groupSize = size
        self.R = R
        self.ns, self.nc = 0, 0

    def get_ni(self):
        for p in self.pop:
            if p.iscoop:
                self.nc += 1
            elif not p.iscoop:
                self.ns += 1

    def get_ri(self):
        """Share of resources for each subset of classes, $r_i$

        Notes
        -----
        Determine r_s and r_c using the provided equation
        """
        self.get_ni()
        nGCs = self.ns * self.GsCs
        nGCc = self.nc * self.GcCc
        SUMj = nGCs + nGCc
        self.rs = self.R * (nGCs / SUMj)
        self.rc = self.R * (nGCc / SUMj)

    def reproduce(self):
        """Determine n_i(t+1) (no individuals added untill whole reproductive cycle determines final numbe
        """
        ns_, nc_ = self.ns, self.nc
        self.ns = ns_ + (self.rs / self.Cs) - (self.K * ns_)
        self.nc = nc_ + (self.rc / self.Cc) - (self.K * nc_)

    def regenerate_pop(self):
        """Regenerate population once reproduction has been accomplished
        """
        pop = []
        for n in range(int(self.ns)):
            pop.append(Gene(self.groupSize, False))
        for n in range(int(self.nc)):
            pop.append(Gene(self.groupSize, True))
        self.pop = pop

    def size(self):
        return len(self.pop)
```

B.4 gene_noext.py

```
    class Gene:
    def __init__(self, issmall:bool, iscoop:bool):
        """Individuals Genotype

        Attributes
        ----------
            joinSize : bool, 0 for small and 1 for large group which the gene should join
            decrease_Gr : bool, True when gene is cooperative (decrease growth rate)
        """
        self.issmall = issmall
        self.iscoop = iscoop
```

C Source Code (Extension)

C.1 main.py

```
from environment import Environment
from gene import Gene

global POPSIZE
# define popsize and generations
POPSIZE = 4000
# t = 4 (reproduce 4x in group before migrant reformation)
epochs = 4


def Initialise(popsize):
    """Initialise population of selfish and cooperative individuals, and small and large prefferences
    Return
    ------
        pop : list of Selfish/Cooperative objects, represents the individuals who are initialised
    """
    pop = []
    for i in range(popsize):
        # Split pop size into 4 genotypes
        if i % 8 in [0,1]:
            # Entries to match ‘isSmall‘ and ‘iscoop‘ for Small and Selfish (SS)
            ind = Gene(True, True)
        elif i % 8 in [2,3]:
            # SC
            ind = Gene(True, False)
        elif i % 8 in [4,5]:
            # LC
            ind = Gene(False, True)
        elif i % 8 in [6,7]:
            # LS
            ind = Gene(False, False)
        pop.append(ind)
    return pop

if __name__ == '__main__':
# 1 Initialisation
    pop = Initialise(POPSIZE)
    env = Environment(pop, POPSIZE, Gene)
    # Set first ratios for plotting
    env.save_ratios()
    for i in range(120):
        print('Iteration ', i)
    # 2 Group Formation
        env.group_formation()
        print('Formed')
    # 3 Reproduction
        env.reproduce()
    # 4 Migrant Pool Formation
        env.migrant_pool_formation()
    # 5 Maintain Global Capacity
        psize = env.get_popsize()
        print(f'Final Pop Size {psize}')
        if psize > POPSIZE: # only rescale if pop-size is too large for environment (fixed at ‘POPSIZE = 4
            env.rescale()
    # Plot Results
    env.plotting()
```

C.2 environment.py

```
import copy
import numpy as np
import group as grp
import random
import matplotlib.pyplot as plt

def filter_pop_by_size(pop: list = []):
    """Filter a pool of individuals into two groups by their preference for large and small groups
    """
    popS, popL = [], []
    for p in pop:
        if p.issmall:
            popS.append(p)
        elif not p.issmall:
            popL.append(p)
    return popS, popL

class Environment:
    t = 4
    def __init__(self, pop, N, Gene):
        self.Gene = Gene
        self.N = N
        self.Groups = []
        self.pool = pop
        self.poolSize = len(pop)
        self.hist_ratios = []
        self.hist_sizes = []

    def get_popsize(self):
        return len(self.pool)

    def group_formation(self):
        """Form Groups as explained in documentation
        """
        self.Groups = []
        popS, popL = filter_pop_by_size(self.pool)
        print(f'Population Size: {len(self.pool)}')
        # Initialise first small group
        tempGroup = grp.Group()
        random.shuffle(popS)
        pS = popS.copy()
        c = 0
        while len(pS) > 0 and c < 3:
            groupSize = tempGroup.size()
            if groupSize < tempGroup.maxSize:
                # randomly shuffle the box of individuals
                random.shuffle(pS)
                # go through all remaining invidiuals until one can join (randomly shuffled order)
                for p in range(len(pS)):
                    pos = -(1+p)
                    indv = pS[pos]
                    # see if we can place individual in group
                    if indv.dist.max > tempGroup.minSize and indv.dist.min < tempGroup.maxSize:
                        tempGroup.add_pop(indv)
                        pS.pop(pos)
                        break # break from loop as individual has been groups
                # If you cant add anymore individuals
                if groupSize == tempGroup.size():
                    # check if group meets the minimum number of prefered individuals to exist
                    if groupSize > tempGroup.minSize:
                        tempGroup.maxSize = 0 # Following iteration, appends group to global stack and beg
                    # if group isnt enough to live, disperse individuals back into pop
                    else:
                        pS = pS + tempGroup.pop
                        tempGroup.reset()
                        c += 1
            else:
                # Append group and initialise through new group
                c = 0
                tempGroup.calc_resource_intake()
                self.Groups.append(copy.deepcopy(tempGroup))
                tempGroup.reset()
                random.shuffle(pS)
                indv = pS[-1]
                tempGroup.add_pop(indv)
                pS.pop(-1)

       # Initialise first Large group
        tempGroupL = grp.Group(issmall=False)
        random.shuffle(popL)
        pL = popL.copy()
        c = 0
        while len(pL) > 0 and c < 3:
            groupSize = tempGroupL.size()
            if groupSize < tempGroupL.maxSize:
                # randomly shuffle the box of individuals
                random.shuffle(pL)
                # go through all remaining invidiuals until one can join (randomly shuffled order)
                for p in range(len(pL)):
                    pos = -(1+p)
                    indv = pL[pos]

                    # see if we can place individual in group
                    if indv.dist.max > tempGroupL.minSize and indv.dist.min < tempGroupL.maxSize:
                        tempGroupL.add_pop(indv)
                        pL.pop(pos)
                        break # break from loop as individual has been groups

                if groupSize == tempGroupL.size():
                    # check if group meets the minimum number of prefered individuals to exist
                    if groupSize > tempGroupL.minSize:

                        tempGroupL.maxSize = 0 # Following iteration, appends group to global stack and be
                    else:
                        pL = pL + tempGroupL.pop
                        tempGroupL.reset(issmall=False)
                        c +=1
            else:
                tempGroupL.calc_resource_intake()
                self.Groups.append(copy.deepcopy(tempGroupL))
                c = 0
                tempGroupL.reset(issmall=False)
                random.shuffle(pL)
                indv = pL[-1]
                tempGroupL.add_pop(indv)
                pL.pop(-1)

        self.save_group_sizes()

    def reproduce(self):
        """Run each Group object's reproduction through ‘t‘ time steps
        """
        Groups_ = []
        for g in self.Groups:
            for i in range(self.t):
                g.get_ri()
                g.reproduce(i)
            g.regenerate_pop()
            Groups_.append(g)

        self.Groups = Groups_
        self.save_ratios()

    def migrant_pool_formation(self):
        """Return all individuals to the migrant pool, ‘self.pool‘
        """
        pool = []
        popSize = 0
        for g in self.Groups:
            for p in g.pop:
                pool.append(p)
                popSize += 1
        self.pool = pool

    def rescale(self):
        """Determine the genotype population split (ratio, ‘fi‘ for i in ss,sc, ls, lc).
        """
        # Init. number of genotypes index: (0) ss, (1) sc, (2) ls, (3) lc
        ni = [0. for i in range(4)]
        self.poolSize = len(self.pool)
        for p in self.pool:
            i = 0
            if p.iscoop:
                i += 1  # index (1) or (3)
            if not p.issmall:
                i += 2  # index (0) or (2)
            ni[i] += 1.
        # Determine the ratio of each genotype
        fi = [ni[i] / self.poolSize for i in range(4)]
        ni_ = [int(fi[i] * self.N) for i in range(4)]
        pool = []
        for i in range(4):
            for j in range(ni_[i]):
                # issmal, iscoop
                if i == 0:
                    pool.append(self.Gene(True, False))
                elif i == 1:
                    pool.append(self.Gene(True, True))
                elif i == 2:
                    pool.append(self.Gene(False, False))
                elif i == 3:
                    pool.append(self.Gene(False, True))
        self.pool = pool

    def save_ratios(self):
        """Save ratios of pool of individuals to plot
        """
        # Init. number of genotypes index: (0) ss, (1) sc, (2) ls, (3) lc
        ni = [0 for i in range(4)]
        self.poolSize = len(self.pool)
        for p in self.pool:
            i = 0
            if p.iscoop:
                i += 1  # index (1) or (3)
            if not p.issmall:
                i += 2  # index (0) or (2)
            ni[i] += 1
        fi = [ni[i] / self.poolSize for i in range(4)]
        self.hist_ratios.append(fi)
        print(f'Pop Ratios: \n    SS {fi[0]}\n    SC {fi[1]}\n    LS {fi[2]}\n    LC {fi[3]}')

    def save_group_sizes(self):
        """Save the frequency of group sizes for plotting
        """
        sizes = np.array([[2],[0]]) # 2D array which holds sizes [idx == 0] and count of sizes [idx == 1]
        for g in self.Groups:
            pop_len = len(g.pop)
            idx = np.where(sizes[0] == pop_len)[0].tolist() # find index of found group size (if exists)
            if idx != []:
                sizes[1,idx] += 1

            else: # if not index found, append size to list and count it as one
                size_ = sizes.tolist()
                size_[0].append(pop_len)
                size_[1].append(1)
                sizes = np.array(size_)
        self.hist_sizes.append(sizes)

    def plotting(self, fignum:int=0):
        """Plot all graphs shown in coursework documentation
        """
        f, sub = plt.subplots(nrows=3)
        ss, sc, ls, lc = [], [], [], []
        selfish, large = [], []

        for r in self.hist_ratios:
            ss.append(r[0])
            sc.append(r[1])
            ls.append(r[2])
            lc.append(r[3])
            selfish.append(r[0]+r[2])
            large.append(r[2]+r[3])

        sub[1].plot(ss, label='Small+Selfish', color='orange')
        sub[1].plot(sc, label='Small+Cooperative', color='lightblue')
        sub[1].plot(ls, label='Large+Selfish', color='orange', linestyle='-.')
        sub[1].plot(lc, label='Large+Cooperative', color='lightblue', linestyle='-.')
        sub[1].axhline(1, color='green', alpha=0.2, linestyle='--')
        sub[1].axhline(0, color='green', alpha=0.2, linestyle='--')
        sub[1].set_xlabel('Generation')
        sub[1].set_ylabel('Population Frequency')
        sub[1].legend()

        sub[0].plot(selfish, label='Selfish', color='lightgrey')
        sub[0].plot(large, label='Large', color='lightgrey', linestyle='-.')
        sub[0].set_xlabel('Generation')
        sub[0].set_ylabel('Population Frequency')
        sub[0].legend()

        arr_ = self.hist_sizes
        sizes = [[0] for i in range(50)]
        for gen in arr_: # within each generation caluclate the ratio of each group size with proportion t
            ratio = (gen[0, :] * gen[1, :]) / np.sum(gen[0, :] * gen[1, :])
            for i in range(len(sizes)): # for each history relating to a group size
                idx = np.where(gen[0] == i)[0].tolist() # return the index of the group size in the genera
                if idx != []:
                    rat_ = ratio[idx].tolist()
                    sizes[i].append(rat_[0])  # append it to a sorted list for group sizes
                else:
                    sizes[i].append(0)
        for s in sizes:
            if any(s): # if there is history for a particular size
```

```
                size_idx = sizes.index(s)
                if size_idx < 10:
                    sub[2].plot(s, label=str(size_idx), color='black', alpha=(((size_idx-2)+2)/13))
                    sub[2].text(len(s)-1, s[-1], f'{str(size_idx)}')
                else:
                    sub[2].plot(s, label=str(size_idx), color='darkblue', alpha=((size_idx-30)/20))
                    sub[2].text(len(s)-1, s[-1], f'{str(size_idx)}')

        sub[2].set_xlabel('Generations')
        sub[2].set_ylabel('Population Frequency')
        sub[2].legend(loc='center left', bbox_to_anchor=(1, 0.5),
                        ncol=2, title='Group Sizes')
        plt.show()
```

C.3 group.py

```
import copy
import random
import numpy as np

class Group:
    # Growth & Consumption Rates (and product)
    Gs, Gc = 0.02, 0.018
    Cs, Cc = 0.2, 0.1
    GsCs, GcCc = Gs * Cs, Gc * Cc
    # Death rate
    K = 0.1

    def __init__(self, pop: list = [], issmall: bool = True, R: float = 0.0):
        self.reset(pop, issmall, R)

    def reset(self, pop: list = [], issmall: bool = True, R: float = 0.0):
        """Initialise group of individuals

                Attributes
                ----------
                    pop : list of Gene, list of individuals
                    issmall : bool, True for small, False for large
                    R : float, resource influx for group
                """
        self.pop = []
        self.prior_coop, self.prior_self = [],[]
        self.popSize = 0
        self.groupSize = issmall
        self.R = R
        # Contain the maximum and minimum sizes a group can take
        if issmall:
            self.maxSize = 10
            self.minSize = 1
        else:
            self.maxSize = 50
            self.minSize = 30
        self.ns, self.nc = 0, 0
        self.prior_ns, self.prior_nc = 0,0



    def add_pop(self, indv):
        """Add an individual to the group while group is forming in ‘environment.group_formation‘
        """
        imax = indv.dist.max
        imin = indv.dist.min
        # Update new bounds for size constraints
        if self.maxSize > imax:
            self.maxSize = imax
        if self.minSize < imin:
            self.minSize = imin
        self.pop.append(indv)
        self.popSize = len(self.pop)

    def calc_resource_intake(self):
        """Equation defined in coursework documentation
        """
        r0 = 0.9425
        n = len(self.pop)
        self.R = n * (r0) * (1.05)**(np.log2(n))


    def get_ni(self):
        """Count the number of selfish and cooperative individuals $n_i$"""
        for p in self.pop:
            if p.iscoop:
                self.nc += 1
            elif not p.iscoop:
                self.ns += 1

    def get_ri(self):
        """Share of resources for each subset of classes, $r_i$
        """
        self.get_ni()
        nGCs = self.ns * self.GsCs
        nGCc = self.nc * self.GcCc
        SUMj = nGCs + nGCc
        self.rs = self.R * (nGCs / SUMj)
        self.rc = self.R * (nGCc / SUMj)

    def reproduce(self, epoch):
        """Determine n_i(t+1)
        """
        if epoch == 0 :
            self.prior_ns, self.prior_nc = self.ns, self.nc
            prior_pop = copy.deepcopy(self.pop)
            for p in prior_pop:
                if p.iscoop:
                    self.prior_coop.append(copy.deepcopy(p))
                elif not p.iscoop:
                    self.prior_self.append(copy.deepcopy(p))
        ns_, nc_ = self.ns, self.nc
        self.ns = ns_ + (self.rs / self.Cs) - (self.K * ns_)
        self.nc = nc_ + (self.rc / self.Cc) - (self.K * nc_)


    def regenerate_pop(self):
        pop_s = copy.deepcopy(self.prior_self)
        pop_c = copy.deepcopy(self.prior_coop)
        ns_diff = int(self.ns - self.prior_ns)
        nc_diff = int(self.nc - self.prior_nc)
        if ns_diff > 0: # if we need to add to population of selfish individuals
            for n in range(ns_diff):
                p = random.choice(self.prior_self)
                pop_s.append(p)
        elif ns_diff < 0:
            for n in range(abs(ns_diff)):
                if len(pop_s) > 0:
                    p = random.choice(pop_s)
                    pop_s.pop(pop_s.index(p))
                else:
                    break
        if nc_diff > 0: # if we need to add to population of selfish individuals
            for n in range(nc_diff):
                p = copy.deepcopy(random.choice(self.prior_coop))
                pop_c.append(p)
        elif nc_diff < 0:
            for n in range(abs(nc_diff)):
                if len(pop_c) > 0:
                    p = random.choice(pop_c)
                    pop_c.pop(pop_c.index(p))
                else:
                    break
        self.pop = pop_s + pop_c

    def size(self):
        return len(self.pop)
```

C.4 gene.py

```
import random
random.seed(7)

class Distribution:
    """Represents the gene for distribution, i.e contains the true preference ‘mu‘ and the margin of error
    """
    # Classify the range which an individual preference is randomly chosen
    LRANGE = [30,50]
    SRANGE = [1,5]
    # Define the std (error rate) of an individual’s size preference
    LSIG = 3
    SSIG = 1

    def __init__(self, pref:bool, mu:float=0.0):
        self.pref = pref
        # First check if mu’s have been given (should be >2 for anny assignment to exist)
        if mu > 0.0 and mu < 10:
            self.mu = mu
            self.sig = self.SSIG
        elif mu > 10:
            self.mu = mu
            self.sig = self.LSIG
        # Small group - set the mu and std of distribution for choice of groups
        elif self.pref:
            self.mu = random.randint(self.SRANGE[0], self.SRANGE[1])
            self.sig = self.SSIG
        # Lagre group
        else:
            self.mu = random.randint(self.LRANGE[0], self.LRANGE[1])
            self.sig = self.LSIG
        self.min = self.mu - self.sig
        self.max = self.mu + self.sig


class Gene:
    def __init__(self, issmall:bool, iscoop:bool, mu:float=0.0):
        """Individuals Genotype

        Attributes
        ----------
            joinSize : bool, 0 for small and 1 for large group which the gene should join
            dist : Distribution, holds ranges and statistical propoerties of individual distribution for p
            decrease_Gr : bool, True when gene is cooperative (decrease growth rate)
        """
        self.issmall = issmall
        self.dist = Distribution(issmall, mu)
        self.iscoop = iscoop
```

C.5 gene_variant.py

```
    import random
import numpy as np

random.seed(7)

class Distribution:
    LRANGE = [[37,43], [37,43], [30,50], [30,50]]
    SRANGE = [[4,5], [2,5], [4,5], [2,5]]
    LSIG = 3
    SSIG = 1

    def __init__(self, pref:bool, mu:float=0.0, ep:int=0):
        self.pref = pref

        # First check if mu's have been given (should be >2 for anny assignment to exist)
        if mu > 0.0 and mu < 10:
            self.mu = mu
            self.sig = self.SSIG
        elif mu > 10:
            self.mu = mu
            self.sig = self.LSIG
        # Small group - set the mu and std of distribution for choice of groups
        elif self.pref:
            self.mu = random.randint(self.SRANGE[ep][0], self.SRANGE[ep][1])
            self.sig = self.SSIG
        # Lagre group
        else:
            self.mu = random.randint(self.LRANGE[ep][0], self.LRANGE[ep][1])
            self.sig = self.LSIG

        self.min = self.mu - self.sig
        self.max = self.mu + self.sig

class Gene0:
    def __init__(self, issmall:bool, iscoop:bool, mu:float=0.0):
        """Individuals Genotype

        Attributes
        ----------
            joinSize : bool, 0 for small and 1 for large group which the gene should join
            dist : Distribution, holds ranges and statistical propoerties of individual distribution for p
            decrease_Gr : bool, True when gene is cooperative (decrease growth rate)
        """
        self.issmall = issmall
        self.dist = Distribution(issmall, mu, ep=0)
        self.iscoop = iscoop

class Gene1:
    def __init__(self, issmall:bool, iscoop:bool, mu:float=0.0):
        """Individuals Genotype

        Attributes
        ----------
            joinSize : bool, 0 for small and 1 for large group which the gene should join
            dist : Distribution, holds ranges and statistical propoerties of individual distribution for p
            decrease_Gr : bool, True when gene is cooperative (decrease growth rate)
        """
        self.issmall = issmall
        self.dist = Distribution(issmall, mu, ep=1)
        self.iscoop = iscoop
class Gene2:
    def __init__(self, issmall:bool, iscoop:bool, mu:float=0.0):
        """Individuals Genotype

        Attributes
        ----------
            joinSize : bool, 0 for small and 1 for large group which the gene should join
            dist : Distribution, holds ranges and statistical propoerties of individual distribution for p
            decrease_Gr : bool, True when gene is cooperative (decrease growth rate)
        """
        self.issmall = issmall
        self.dist = Distribution(issmall, mu, ep=2)
        self.iscoop = iscoop
class Gene3:
    def __init__(self, issmall:bool, iscoop:bool, mu:float=0.0):
        """Individuals Genotype

        Attributes
        ----------
            joinSize : bool, 0 for small and 1 for large group which the gene should join
            dist : Distribution, holds ranges and statistical propoerties of individual distribution for p
            decrease_Gr : bool, True when gene is cooperative (decrease growth rate)
        """
        self.issmall = issmall
        self.dist = Distribution(issmall, mu, ep=3)
        self.iscoop = iscoop
```