

THE INFLUENCE OF FAMILIARITY, RESOURCES, AND SAMPLING ON SOCIAL STRUCTURE: A SIMULATION-BASED STUDY

A thesis submitted in partial fulfilment of the degree of
Masters in Science

by

Anvitha S.



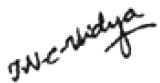
Jawaharlal Nehru Centre for Advanced Scientific Research,
Bengaluru 560064, India.

March 2022

CERTIFICATE

This is to certify that the work presented in this thesis titled *The Influence of Familiarity, Resources, and Sampling on Social Structure: A Simulation-based Study* has been carried out by Ms. Anvitha S. under my supervision at the Evolutionary and Integrative Biology Unit (formerly Evolutionary and Organismal Biology Unit), Jawaharlal Nehru Centre for Advanced Scientific Research, Bengaluru, India, and that the results in this thesis have not previously formed the basis for the award of any other degree, diploma, or fellowship.

Date: 9 March 2022


Prof. T.N.C. Vidya

DECLARATION

I declare that the matter presented in my thesis titled *The Influence of Familiarity, Resources, and Sampling on Social Structure: A Simulation-based Study* is the result of studies carried out by me at the Evolutionary and Integrative Biology Unit (formerly Evolutionary and Organismal Biology Unit of the Jawaharlal Nehru Centre for Advanced Scientific Research, Bangalore, India, under the supervision of Prof. T.N.C. Vidya, and that this work has not been submitted elsewhere for any other degree.

In keeping with the general practice of reporting scientific observations, due acknowledgement has been made wherever the work described has been based on the findings of other investigators. Any omission, which might have occurred by oversight, is regretted.

Place: Bengaluru

Anvitha S
Anvitha S.

Date: 9 March 2022

ACKNOWLEDGEMENTS

I thank JNCASR for the fellowship and logistics to conduct this project.

I am grateful to my advisor Prof. TNC Vidya for providing me with the opportunity to work on this project. She has been highly supportive, insightful, and encouraging throughout my course work and while working on this project.

I am also thankful to Prof. Amitabh Joshi for being a fantastic teacher. Discussions with him have always been very interesting.

I am thankful to Satya and Keerthi for their unwavering support and feedback, especially when I was confused and unsure of things. I also thank Manan and Anuj.

I thank Manu for helping me figure out how to run certain C++ codes on Matlab.

I thank past and present members of ABL and EBL for the discussions and happy distractions.

I am grateful to my sister, Anushree, for always being there. I am thankful to my family and friends for their counsel, patience, and love.

Contents

Certificate	iii
Declaration	v
Acknowledgements	vii
Thesis Abstract	1
Chapter 1: Introduction	3
Chapter 2: Methods	7
Chapter 3: Results	17
Chapter 4: Discussion	37
References	41
Supplementary information	45

THESIS ABSTRACT

The wide diversity in social structures is thought to be the result of selection for individual behavioural strategies and inter-individual interaction patterns that maximise fitness under different environments (van Schaik and van Hoff 1983, Kappeler and van Schaik 2002). Recently, several simulation studies have attempted to generate social structure through simple and general models (Ilany and Akçay 2015, Rios and Kraenkel 2017, Cantor and Farine 2018) but have not simultaneously examined the effect of resource conditions on social structure. Here, I describe a simulation study that I conducted to examine whether a social structure emerged when individuals associated with others to different extents based on familiarity, in a habitat with limited and patchy ephemeral resources that either varied in quantity or stayed constant over time. Additionally, I also examined the sampling conditions under which a social structure would be wrongly inferred even when associations were random. Thus, the first question dealt with possible social structure emerging due to simple rules, whereas the second dealt with possible apparent social structure when there was none. Results from my simulations showed that when resources were limited and ephemeral, social structure, characterised by low network density and high modularity, emerged only when all the associations were with familiar individuals and there was no temporal variation in resources. When there was temporal variation in resources, this structure broke down even when many associations were with familiar individuals, becoming similar to that obtained for random association. When associations were only with familiar individuals, social structure could also emerge if resources were not limited. Simulations to address the second question showed that low sampling intensity and a small sampling period could lead to apparent social structure (with high modularity and low density) even in a population with random associations. Moderately intense sampling conducted over long periods of time was essential to detect social structure close to the true structure.

References

1. Cantor M and Farine DR. (2018). Simple foraging rules in competitive environments can generate socially structured populations. *Ecology and Evolution*, 8(10), 4978–4991.
2. Ilany A and Akçay E. (2016). Social inheritance can explain the structure of animal social

networks. *Nature Communications*, 7(1), 1–10.

3. Kappeler PM and van Schaik CP. (2002). Evolution of primate social systems. *International Journal of Primatology*, 23(4), 707–740.
4. Rios VP and Kraenkel RA. (2017). Do I know you? How individual recognition affects group formation and structure. *PLoS One*, 12(1), e0170737.
5. van Schaik CP and JARAM Van Hooff. (1983). On the ultimate causes of primate social systems. *Behaviour*, 85, 91–117.

CHAPTER 1: INTRODUCTION

Group-living is a common trait among animals, and is thought to be selected for when the benefits of being in groups outweigh the costs (Alexander 1974, Krause and Ruxton 2002). The benefits of group living include protection from predation and infanticide, better access to abiotic resources, increased opportunities to mate, and cooperative offspring care, while the major costs include competition for resources and mates, and increased transmission of pathogens/parasites (Janson and van Schaik 1988, Dehn 1990, Wrangham *et al.* 1993, Gompper 1996, Sterck *et al.* 1997, Hass and Valenzuela 2002, Altizer *et al.* 2003, Silk 2007, Clutton-Brock and Huchard 2013). Group living animals exhibit a wide diversity of social structures, which refer to the content, quality, and patterning of social interactions and the relationships that result from repeated interactions amongst the members of the population (see Hinde 1976, Kappeler and van Schaik 2002). Specific social structures are thought to be adaptive evolutionary outcomes of the ecological factors selecting for individual behavioural strategies and inter-individual interaction patterns that maximise survival and reproductive success (van Schaik and van Hoff 1983, Kappeler and van Schaik 2002). The social structure in a population can also, in turn, affect individual reproductive success, gene flow and information flow within a population, and population dynamics (Whitehead 2008a, Wilson 1975).

In primates, the socio-ecological model was proposed to explain how resource-risk distributions select for specific female behavioural strategies, which lead to diverse social structures across different species (Wrangham 1980, van Schaik 1989, Sterck *et al.* 1997). According to this model, when resources are abundant and predation risk exists, females live in egalitarian groups whose composition may change with time due to dispersal. However, when resources are limited, females are resident in natal groups, within which the nature of relationships amongst females may range from egalitarian to despotic and nepotistic depending on the extent of food competition. There has been variable support for this model in several primate species (Whitten 1983, van Schaik *et al.* 1983, Fashing 2001, Grueter *et al.* 2016; see Thierry 2008, Koenig and Borries 2008, and Clutton-Brock and Janson 2012 for reviews), and, recently, there have been calls to consider models of social structure that are more simple, general, and generative (Ilany and Akçay 2015, Cantor and Farine 2018, Firth *et al.* 2017, He *et al.* 2019). To that end, Ilany and Akçay (2015) constructed a social inheritance model in which individuals tended to associate with associates of their parents.

This model generated aspects of social structure that resembled those of spotted hyena, rock hyrax, bottlenose dolphin, and sleepy lizard. Another model by Cantor and Farine (2018) tracked individuals who competed for a single resource patch, and continued to forage with those with whom they foraged last only if that endeavour had resulted in the acquisition of sufficient food. This model produced kin-structured groups with stable group compositions. The authors further claimed that this model provided a simple mechanism for the emergence of foraging specialisation. An agent-based model by Rios and Kraenkel (2017) found highly modular and spatially segregated groups when individuals moved closer or away from one another depending on the memory of previous interactions. In contrast to the previous studies, which only investigated the effects of simple individual rules of association on social structure, in this present study, I aimed to find out if such simple rules of association are by themselves sufficient to explain social structure, irrespective of resource distributions. To elaborate, I examined how resource constraints, resource variation, and the simple behavioural strategy to associate with familiar individuals affected social relationships and the social structure that resulted from them. I used individual-based simulations, which can be thought of as *in silico* experiments that aim to generate organisation or structure occurring at a higher level through the actions and interactions of entities at a lower level in lieu of actual experimentation (Bodine *et al.* 2020). Thus, these simulations would be helpful in understanding whether simple local rules could result in the emergence of higher-order phenomena or patterns such as social structure, quantified here in terms of association patterns and network measures (see below).

I also wanted to examine whether sampling itself could result in the detection of social structure when there was none. Various measurable attributes such as rates of interactions/associations, asymmetry in interactions, linearity of dominance, social differentiation, and stability of associations have been used to describe social structure (Whitehead 2008a). Network analysis and network measures such as degree, density, modularity, clustering coefficient, and path length have also been used to describe social structure/social network structure (for example, Lusseau and Newmann 2004, Wolf *et al.* 2007, Sundaresan *et al.* 2007, Cantor *et al.* 2012, Nandini *et al.* 2018). To measure these different attributes of social structure, data are often collected on the interactions or associations occurring amongst identified individuals in a population. However, since every interaction and every individual cannot be sampled, it is possible that the estimates of social structure may not reflect the actual social structure (as also previously noted, for example, by

Whitehead 2008b, Franks *et al.* 2010, Voelkl *et al.* 2011). I, therefore, also used individual-based simulations to examine to what extent social structure might be wrongly inferred as an artefact of low sampling even when individuals associated with one another randomly.

The specific questions I addressed were the following:

1. *Does social structure emerge under different resource conditions when individuals associate based on familiarity?*

If simple association rules or individual behavioural strategies are solely responsible for patterns in group composition, the social structure that resulted from it would not be very different under different resource conditions. To examine the influence of simple association rules and resource conditions on social structure, I considered models wherein individuals associated with others to different extents based on familiarity in a habitat with limited and patchy ephemeral resources that either varied or not in quantity over time. If individuals showed no preference in their associates, i.e., if associations were random, groups were expected to be random subsets of individuals in the population, the compositions of which would change with time. This would result in a social structure characterised by low modularity, high density, absence of preferential associations, and the absence of temporal stability in associations. However, if individuals showed some inertia in whom they associated with, based on their past memory of associations (solely familiarity, and not positive or negative effects of interactions), I expected to find stable group compositions with time. This would result in a social structure characterised by high modularity, low density, presence of preferential associations, and temporal stability in associations. The above expectations of social structure would not be different across differently-sized resource patches or different types of temporal variation in resource quantity (presence or absence of variation) if resource conditions did not influence social structure. However, if resources were limiting, individuals would be forced to occupy any free patch they could find, resulting in them not being able to always maintaining their preferred associations. This could lead to the lack of social structure if resources were varying spatiotemporally. In my models, I used such a scenario in which resources were limited, as might often be the case in the wild.

2. *Under what conditions of inadequate sampling would social structure be wrongly inferred when individuals actually associate with one another randomly?*

If individuals associated with one another randomly and if the observer did not have sufficient data on all the individuals and their associations, it might happen just by chance that some

associations were recorded to occur more, or less, frequently than others. The population would then appear to have preferential associations, and social structure attributes measured would not indicate random association. Therefore, if large amounts of data on individuals and their associations were not obtained, i.e., if a large number of groups were not sampled for long periods of time, social structure attributes measured, and the inferences made using them might be very different from the true attributes. In order to examine how sampling affects inferences of social structure even when associations are random, I modelled scenarios of sampling from a population with random associations by considering different sampling intensities and different lengths of sampling period. Additionally, I also examined if the average group size affected such inferences. I expected to find populations with high modularity and low density when sampling intensity was low and sampling period length was short, but not when sampling intensity was high and sampling period length was long. I also expected to find populations with modularity and density closer to the true value when group sizes were larger than when group sizes were smaller simply because more associations would be sampled when groups were larger than when groups were smaller, for the same sampling effort.

CHAPTER 2: METHODS

As mentioned above, I used individual-based simulations to address both questions. The details of the simulations are given below.

2.1 Does social structure emerge under different resource conditions when individuals associate based on familiarity?

2.1.1 Simulation overview

I considered a static population of fixed size (200 individuals), within which individuals associated with one another based on specified rules of association and resource conditions (see below). No assumption was made about the age or sex of the individuals or the nature of the associations amongst individuals. I generated a patch size distribution at every time step, and assigned individuals to the patches based on familiarity with one another. I assumed individuals within a patch to be associating with one another, and used the association data to measure different attributes of social structure.

Resource conditions

A patch was defined as an area with finite resources such that a patch of size 1 had one available (feeding) site, which could, therefore, support only one individual, and so on. I considered two kinds of resource distributions. In the first kind, there was no variation in the amount of resource available either within a time step or across time steps, i.e., all the patches in a given time step had equal patch size, and the size did not change with time. This represented ephemeral resources that were regenerated every time step with the same quantity (although unlikely to be seen in the wild). In the second kind, there was variation in the amount of resource available within a time step and across time steps, i.e., patch sizes varied across the patches present at a given time step and across time steps. This represented ephemeral resources that were regenerated every time step in different quantities. I coded the presence or absence of temporal variation in resources using a patch size variation parameter (value=1 when variation present, 0 when absent). Additionally, I examined the effect of resource constraints by considering average patch size as a parameter. Average patch size was either 2 or 4; if it was 2, patches could accommodate two individuals on average, whereas, if it was 4, patches could accommodate four individuals on average. At each time step, I generated the patch size distribution either by setting all the patch sizes to the average patch

size (in the case of no patch size variation), or by obtaining patch sizes from a zero-truncated negative binomial distribution with the appropriate mean value (group sizes: Cohen 1972, Caraco 1978) (when there was patch size variation). In all the simulations, the sum of patch sizes of all the patches was equal to the population size. This represented a habitat with just enough resources for all the individuals in the population.

Familiarity-based association

In all the simulations, I used two parameters, memory and memory length, to implement familiarity-based associations. The memory parameter determined how often associations in the population were preferential associations based on familiarity; in other words, how frequently did individuals associate with their previous associates. I used memory parameter values of 1, 0.5, or 0. If the memory parameter was 1, 100% of the associations were based on familiarity. If the memory parameter was 0.5, 50% of the associations were based on familiarity. If the memory parameter was 0, there were no preferential associations based on familiarity, and all the associations were random. The memory length parameter determined how long the memory of familiarity lasted; it was the farthest time step in the past, from which a given individual remembered its previous associates. I used memory length parameter values of 1 (short; individuals remembered previous associates from only 1 time step back), 20 (medium), or 400 (long). The values of the above two parameters determined the extent to which associations were based on familiarity.

2.1.2 Generating associations

I generated a patch size distribution (determined by the average patch size and the patch size variation parameters) in each time step such that the sum of patch sizes of all the patches was equal to the population size. In the first time step, individuals were assigned to the feeding sites within patches randomly (Figure 1). From the second time step onwards, associations were based on familiarity to different extents (due to memory and memory length parameters). Since the total amount of resource available was equal to the population size, all the sites in each patch would be occupied. Due to resource constraints, group size in a patch could not be larger than the patch size (number of sites available in a patch). From the second time step onwards, I selected the first individual for a given patch randomly from the population. Then, I drew a uniform random number between 0 and 1. If this number was less than the memory parameter, I created an association based on familiarity by adding an individual from the pool of previous associates of the first individual to the patch. The available pool of previous

associates of the first individual was based on the memory length parameter, which determined the farthest time step in the past from which the first individual remembered its previous associates. If the first individual had multiple previous associates, I selected a previous associate based on weighted probabilities (weighted by the number of times each associate had previously associated with the first individual). If the first individual did not have any available previous associate or did not remember any previous associate, the second individual assigned to the patch was an individual unknown to the first individual, selected randomly from the pool of remaining individuals in the population, in order to fill the patch. If the random number drawn was larger than the memory parameter, I created a non-preferential association by adding a random individual from the population as the second individual of the patch.

I added individuals one at a time to the patch in the manner above until all the feeding sites within a patch were occupied, i.e., the number of individuals in the patch was equal to the patch size allocated. When more than two individuals had already been assigned to a patch and the next association was to be based on familiarity, the next individual was selected with a probability weighted by the average of the number of times it had previously associated with the individuals already assigned to the patch. Thus, it could so happen that the next individual chosen might have been a previous associate of only one/some and not all the individuals in the patch. I added individuals to each patch sequentially until all the individuals in the population were assigned to patches. As mentioned above, in all the simulations, all the individuals present within a patch constituted a group and were considered to be associating with one another. Since individuals were assigned to patches until the patches were filled, group size distribution was the same as the patch size distribution.

While adding individuals to patches, I ensured that the same individual was not allotted to more than one patch in the same time step. I repeated the process of obtaining a patch size distribution and the sequential addition of individuals to patches every time step for 400 time steps. The group composition of each group at a time step was considered one sighting, and the group compositions of all the groups in that time step constituted the sighting data for that time step.

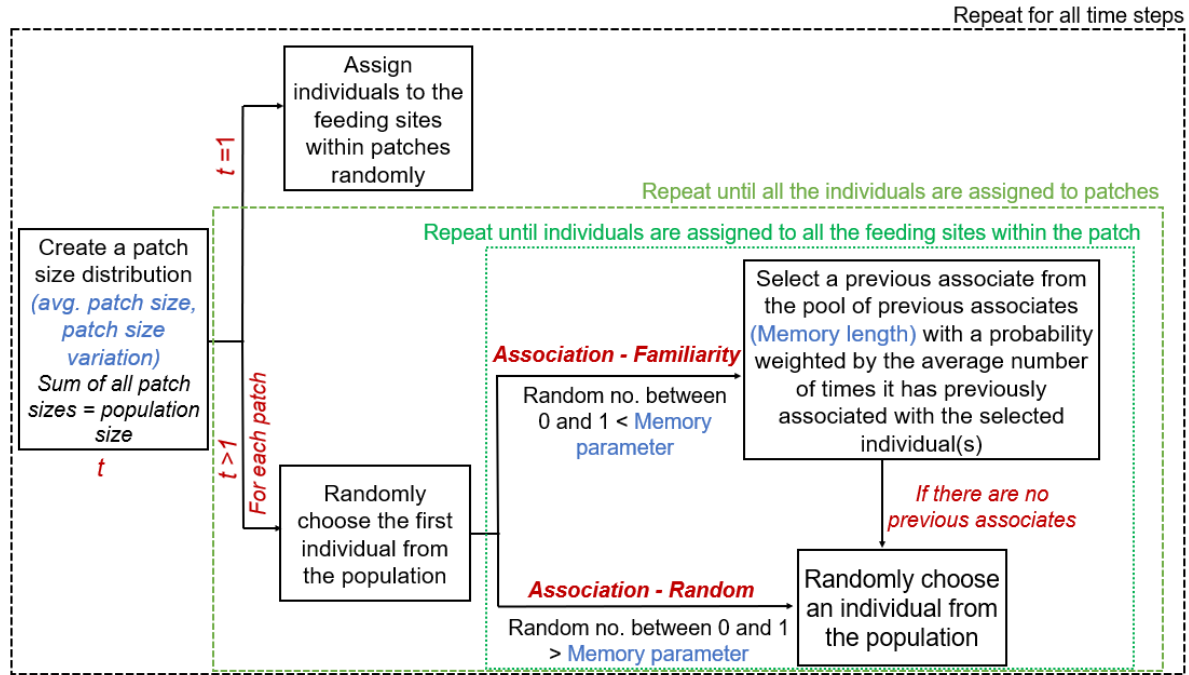


Figure 1. Flow chart of the simulations conducted to examine the effect of memory, memory length, average patch size, and patch size variation on social structure.

2.1.3 Detecting social structure

Once the sighting data were generated, I used the following four aspects to look for social structure: density, modularity, stability of associations, and social differentiation.

Calculating density and modularity

I excluded group composition data from the first twenty time steps, i.e., all the groups found in the first twenty time steps were removed from the sighting data. I then calculated an association index (AI) matrix using the remaining sighting data. The AI matrix contained pairwise association indices, measuring the extent of association between pairs of individuals. I used the simple ratio index (Cairns and Schwager 1987) to calculate AI as $AI = \frac{N_{AB}}{N_A + N_B - N_{AB}}$, where N_{AB} was the number of times A and B were seen together, N_A was the total number of sightings of A (alone and with others), and N_B was the total number of sightings of B. Using the AI matrix, I constructed a weighted network, in which each node represented an individual, each edge represented the association between two individuals, and AI was the weight of the edge between the two individuals. I calculated density and modularity of this weighted network. Density is a measure of the extent of connectivity in the network, and is calculated as the ratio of the observed number of edges to the number of

possible edges in the network (see Wasserman and Faust 1994). Thus, a density close to zero indicates a poorly connected network with very few ties, whereas a value of 1 indicates a fully connected network with all possible ties.

Modularity measures the strength of the division of a network into communities. A network would have distinct communities when modularity is high, which is when the number and strength of connections between the nodes within communities are higher than that expected by random chance (Newman 2004). In all the simulations, the modularity of the network was obtained using the Louvain algorithm (Blondel *et al.* 2008). The Louvain algorithm is a heuristic method of determining community structure hierarchically by optimising Girvan-Newman modularity given by the formula (Newman 2004):

$$Q = \frac{1}{2m} \sum_{ij} [A_{ij} - \frac{k_i k_j}{2m}] \delta(c_i, c_j)$$

Where, A_{ij} is the weight of the edge between i and j

$k_i = \sum_j A_{ij}$ is the sum of weights of the edges attached to node i

c_i is the identity of the community to which node i is assigned

$\delta(c_i, c_j)$ is 1 if nodes i and j are assigned to the same community and 0 otherwise

$$m = \frac{1}{2} \sum_{ij} A_{ij}$$

Girvan-Newman modularity ranges from -1 to 1. A value of 0 indicates that the strength of connections within communities is not different from random chance, while a value greater than 0 indicates that the strength of connections within communities is greater than that expected by random chance. Usually, a network is considered to be partitioned into meaningful communities when the modularity value is greater than 0.3 (Newman 2004).

The Louvain algorithm has two main phases (together called a pass), which were repeated iteratively. In phase one, each node was assigned to a different community. Then for a given node i , the change in modularity was calculated if i was moved from its original community to the community of its neighbour j . The community assignment of node i was changed only if there was a gain in modularity. The process was repeated sequentially for all nodes of the network until there was no change in modularity, i.e., when a local maximum was reached. In phase two, a new network was created in which the nodes of the network were the communities found in phase one. The weights of the edges between two new nodes (each being a community containing more than one node in phase 1) were determined by adding

the weights of edges from every node in one community to another, with the edges between nodes of the same community forming self-loops in the new network. Phases one and two were repeated iteratively until there was no change in modularity. Thus, with each pass, a higher level of hierarchy in the community structure, if present, could be detected. The Louvain algorithm was implemented in MATLAB R2019b (The MathWorks, Inc., 1994-2022, www.mathworks.com; code appended at the end of the thesis) by running the C++ executables (generated through Cygwin, a software that compiles and executes source codes written to run on Unix-like operating systems, available at <https://cygwin.com/>.) for the codes provided by the authors at <https://sourceforge.net/projects/louvain/>. The Louvain algorithm took the list of pairwise association indices as input, detected communities, and provided the number of passes, the number of communities produced in each pass and the identity of communities that a given individual was assigned to in each pass as outputs. I calculated maximum modularity reached in the final pass using the information on the number of communities detected and the composition of each community.

Stability of associations and social differentiation

To understand how associations changed with time, the stability of associations and social differentiation were measured at every time step. The stability of associations was examined by plotting the average cumulative number of new associates per individual against time. For each individual in the population, the cumulative number of new associates it had at time step t was obtained by comparing its associates (group members) at time step t with associates from time step 1 to $t-1$. This was then averaged across all the individuals in the population. Social differentiation is the variation in the dyadic probabilities of association calculated through the coefficient of variation (CV) of pairwise AIs (Whitehead 2008a). If there were preferred/avoided associates, this CV of AIs would be higher than that expected by random chance. The presence of preferential associations would also lead to a network with distinct communities and high modularity. To determine the CV of AI at time step t , first, an AI matrix was constructed using all the sighting data from time step 1 to t . Then, the CV was calculated for all the elements of the AI matrix. The average cumulative number of new associates per individual and the CV of AI was calculated at every time step for 400 time steps.

If individuals in the population continued to associate with the same set of individuals, i.e., if there was stability in associations, the average cumulative number of new associates per individual would not change with time, the density of the network would be low, and the

modularity of the network would be high. Similarly, if individuals had consistent and strong preferences in their associations, the CV of pairwise AIs would be constant and higher than that expected if associations were random across all time steps. Further, the density of the network would be low, and modularity would be high.

I ran simulations to examine the effects of memory, memory length, the average patch size, and patch size variation parameters (see Table 1 for parameter values) on network density, network modularity, stability of associations, and social differentiation. I used 36 simulations (Supplementary material, Table 1) to explore all combinations of the above factors, resulting in a fully factorial design. Each simulation was run ten times. All the simulations were run on MATLAB R2019b (The MathWorks, Inc, 1994-2022, www.mathworks.com; code appended at the end of the thesis).

Table 1. Initialisation of different parameters for simulations conducted to examine the effect of memory, memory length, average patch size, and patch size variation on social structure.

Parameter	Value
Population size	200
Total time	400 time steps
Memory parameter	1, 0.5, 0
Memory length	1, 20, 400
Average patch size	2, 4
Patch size variation	Present, Absent

2.1.5 Data analysis

To determine the effect of memory, memory length, average patch size, and patch size variation on density and modularity, I ran two four-factor ANOVAs using Statistica 7 (StatSoft, Inc. 2004). In these ANOVAs, memory, memory length, average patch size, and patch size variation were treated as fixed factors. There were 10 replicates for each level of the factors. I did not perform ANOVAs on the cumulative number of new associates or the CV of AI because they were expected to give the same patterns as density and modularity, respectively. I calculated effect sizes using eta-square.

2.2 Under what conditions of inadequate sampling would social structure be wrongly inferred when individuals actually associate with one another randomly?

2.2.1 Simulation overview

I used a static population of 200 individuals again. At every time step (for 400 time steps in all), I obtained a group size distribution, and assigned individuals to the groups randomly. To simulate field sampling, I noted the group compositions of a set of groups at different time steps (see below), and each group was considered a sighting. To examine the effects of sampling on inference of social structure, two parameters were considered, namely, sampling intensity and sampling period length. Sampling intensity determined how often groups were sampled (sampling interval) and the proportion of groups that were sampled. Sampling period length determined the total time for which sampling was done to infer social structure. The simulations also examined if sampling effects were different for different average group sizes.

2.2.2 Generating associations and sighting data

I obtained a group size distribution every time step by drawing numbers from a zero-truncated negative binomial distribution with a specific average group size (the average group size parameter was either 2 or 4 depending on the simulation). I ensured that the group sizes added up to the population size. I then randomly assigned all the individuals within the population to different groups, and all the individuals within a group were considered to be associating with one another. Sampling intensity was determined by the proportion of groups sampled and the sampling interval parameters. Depending on the proportion of groups sampled parameter (0.2, 0.4, 0.6, 0.8, or 1; a value of 0.2 corresponded to 20% of all the groups present at a given time step being sampled and so on), a certain percentage of groups created at a given time step was sampled randomly, i.e., group compositions of only those randomly chosen groups constituted the sighting data for that time step. Groups were sampled every other time step, except for the sampling intensity of 1, when they were sampled every time step, in order to obtain the true values. Depending on the sampling period length parameter (20, 100 or 400), groups were sampled for the given number of time steps, and the sighting data for that period were collated together to examine different attributes of network structure (see Figure 2). If the sampling period length parameter value was smaller than the total number of time steps in the simulation, sighting data were obtained for more than one sampling period, and the attributes of network structure were calculated for each. For example, since the total number of time steps was 400, when the sampling period length was

20 or 100, sighting data for 20 different short sampling periods or four different moderately long sampling periods were obtained.

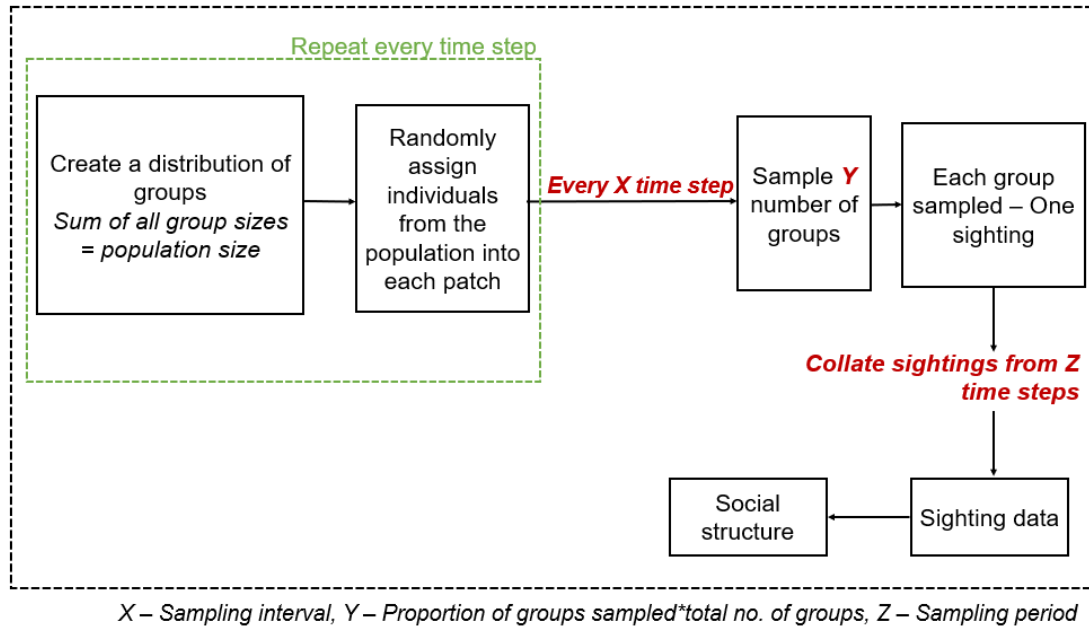


Figure 2. Flow chart of simulations conducted to examine the effect of sampling on social network structure.

2.2.3 Detecting social structure

Once the sighting data for a given sampling period were generated, the data were further filtered to retain only those individuals that were sighted at least two times. It is a common practice amongst researchers to filter out poorly sampled individuals to reduce the bias in network estimates. I used the modified sighting data to obtain an association index (AI) matrix and construct a weighted network. I then calculated the density and modularity of the weighted network in the same manner as in the previous simulations.

Table 2. Initialisation of different parameters for simulations conducted to examine the effect of sampling on social network structure.

Parameter	Value
Population size	200
Total time	400 time steps
Average group size	2, 4
Proportion of groups sampled	0.2, 0.4, 0.6, 0.8, 1
Sampling period length	20, 100, 400

2.2.5 Data analysis

I set up 30 simulations (see Supplementary material, Table 2) to explore all the combinations of sampling intensity, sampling period length, and average group size parameters, resulting in a fully factorial design (Table 2). Each simulation was run ten times on MATLAB R2019b (The MathWorks, Inc, 1994-2022, www.mathworks.com; code appended at the end of the thesis). I ran two three-factor ANOVAs using Statistica 7 (StatSoft, Inc. 2004), with modularity and density as the dependent variables, and sampling intensity, sampling period length, and average group size as fixed factors.

CHAPTER 3: RESULTS

3.1 Does social structure emerge under different resource conditions when individuals associate based on familiarity?

3.1.1 Network density

I found significant main effects of memory, memory length, average patch size, and patch size variation, as well as significant interaction effects of the two-way, three-way, and four-way interactions on the density of the network (Table 3). However, while the effect sizes for the main effects of memory ($\eta^2=0.379$) and patch size variation ($\eta^2=0.279$), and the interaction between memory and patch size variation ($\eta^2=0.293$) were large, the effect sizes for the main effect of average patch size and the three-way interaction of memory, average patch size, and patch size variation were weak (Table 3). The effect sizes for the main effect of memory length, all the other interactions, and the error were close to zero (Table 3).

Density was significantly higher (0.957) when there was no role of memory (memory=0) than when associations were completely based on memory (memory=1; density=0.468; Tukey's HSD test: $P<0.001$) (Figure 3a). Although, density at a memory of 0.5 (0.902) was similar to that at a memory of 0 (0.957; Figure 3a), the post hoc tests showed a significant difference ($P<0.001$). As mentioned above, memory length had a negligible effect size, but post-hoc tests showed a significant difference in density between memory length of 1 (0.786) and 20 (0.770; Tukey's HSD test: $P<0.001$), and memory length of 1 (0.786) and 400 (0.770; $P<0.001$), whereas there was no significant difference between memory length of 20 and 400 ($P=0.756$; Figure 3b). The main effect of average patch size was also low, accounting for only 2.6% of the variation in density, with density being higher (0.832) when the average patch size was 4 than when the average patch size was 2 (0.719) (Figure 3c). Patch size variation accounted for 27.9% of the variation in density, with density being higher when patch size variation was present (0.963) than when it was absent (0.588) (Figure 3d).

Table 3. Results of four-factor ANOVA with network density as the dependent variable and memory, memory length, patch size variation, and average patch size as the fixed independent variables. The three large effect sizes are marked in bold.

Effect	<i>SS</i>	<i>df</i>	<i>MS</i>	<i>F</i>	<i>P</i>	η^2
{1}Memory	17.200	2	8.600	2656689.309	<0.001	0.379
{2}Memory length	0.021	2	0.010	3170.922	<0.001	0.000
{3}Average patch size	1.158	1	1.158	357766.426	<0.001	0.026
{4}Patch size variation	12.652	1	12.652	3908402.764	<0.001	0.279
Memory*Memory length	0.010	4	0.003	801.942	<0.001	0.000
Memory*Average patch size	0.214	2	0.107	33121.798	<0.001	0.005
Memory length*Average patch size	0.024	2	0.012	3778.170	<0.001	0.001
Memory*Patch size variation	13.280	2	6.640	2051185.461	<0.001	0.293
Memory length*Patch size variation	0.000	2	0.000	18.593	<0.001	0.000
Average patch size*Patch size variation	0.167	1	0.167	51435.344	<0.001	0.004
Memory*Memory length*Average patch size	0.012	4	0.003	946.630	<0.001	0.000
Memory*Memory length*Patch size variation	0.024	4	0.006	1819.557	<0.001	0.001
Memory*Average patch size*Patch size variation	0.597	2	0.298	92195.890	<0.001	0.013
Memory length*Average patch size*Patch size variation	0.000	2	0.000	42.687	<0.001	0.000
1*2*3*4	0.029	4	0.007	2207.686	<0.001	0.001
Error	0.001	324	0.000			0.000

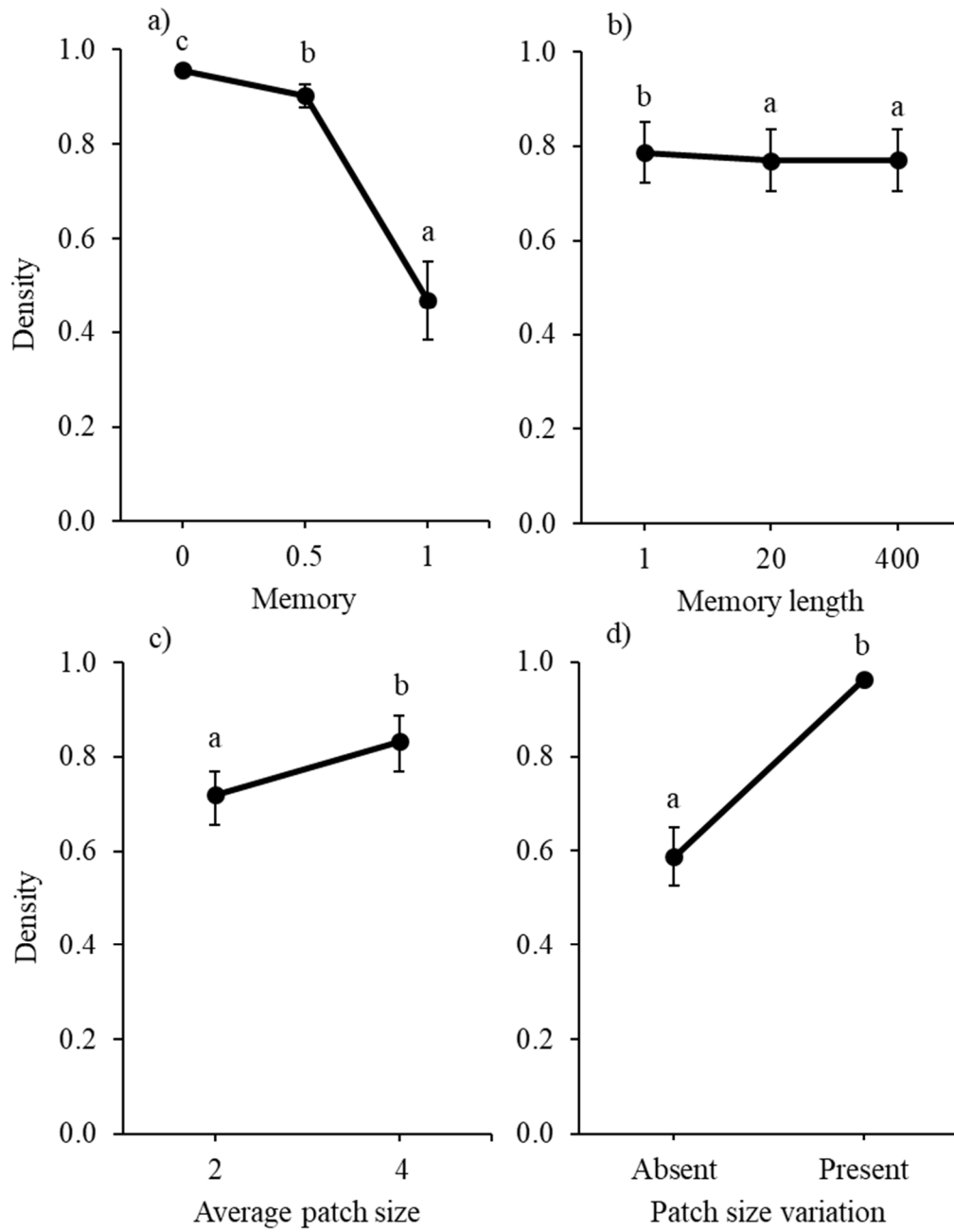


Figure 3. The main effects of memory, memory length, average patch size, and patch size variation on network density. Shared alphabets do not indicate significant difference, while different alphabets do, with $c > b > a$. The error bars are 95% CI about the mean.

The presence of patch size variation increased density (compared to the absence of patch size variation) to different extents depending on the value of memory. All the pairwise comparisons for the interaction between memory and patch size variation were statistically significant. Density at memory of 0 was statistically significantly higher (Tukey’s HSD test: $P<0.001$) but not very different in value when patch size variation was present (0.989) or absent (0.925; Figures 4, 5). Density at memory of 0.5 was somewhat higher ($P<0.001$) when patch size variation was present (0.975) than when it was absent (0.830), while the greatest difference in density was seen at a memory of 1, with the density being starkly higher when patch size variation was present (0.926) than when it was absent (0.010, $P<0.001$; Figures 4, 5). This was because when patch sizes varied, some associations had to be made with unknown individuals whenever previous associates were unavailable, leading to increased network connectivity (see below). On the other hand, when there was no patch size variation, associations with unknown individuals were not made when there was a memory of 1 (leading to very low density), whereas associations were made randomly half the time when memory was 0.5, and associations were made randomly all the time when memory was 0 (leading to high connectivity and hence density). This is seen in the number of new associates below.

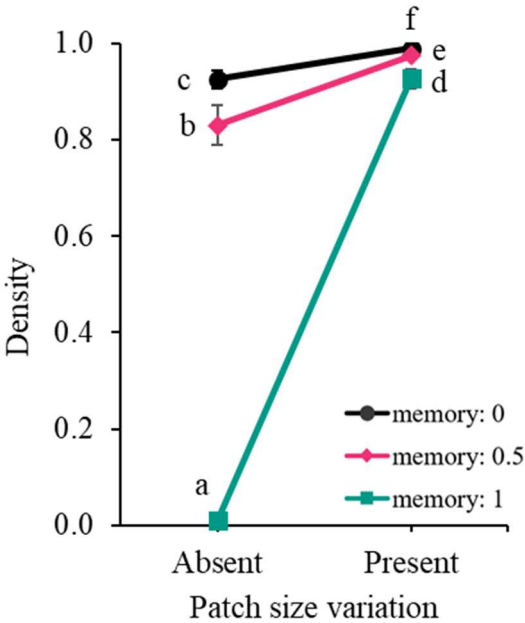


Figure 4. The interaction effect between memory and patch size variation on network density. When memory was 1 and patch size variation was absent, density was 0.010. Different alphabets indicate significant difference, with $f>e>d>c>b>a$. The error bars are 95% CI about the mean.

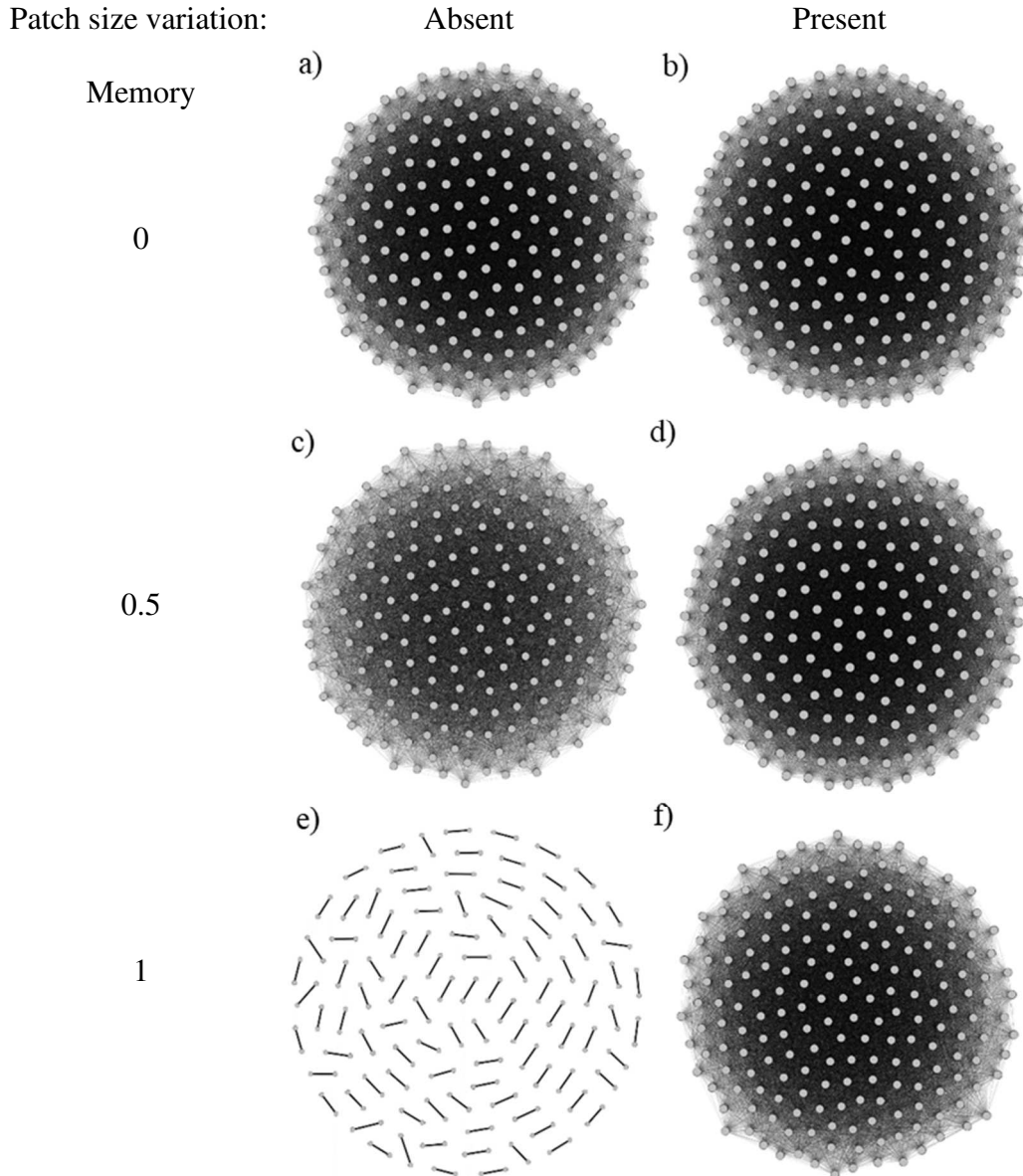


Figure 5. Association networks across different memory and patch size variation values constructed for simulations in which average patch size was 2 and memory length was 400. These networks were constructed using sighting data from the 21st to the 400th time steps of one of the 10 runs of each simulation. Association networks for average patch size of 4 and memory lengths of 1 and 20 were similar to the above networks (not shown).

3.1.1.1 Stability in associations: Average cumulative number of new associates per individual

When memory was 0 or 0.5, since all or some associations were random, individuals met new associates over time (black and pink lines corresponding to memory of 0 and 0.5, respectively, in Figure 6, Supplementary Figures 1, 2), resulting in the average cumulative number of new

associates per individual increasing over time. This would result in a highly connected network with high density as I found above. When memory was 1, individuals gained new associates with time when patch size variation was present (green lines corresponding to memory of 1 in Figure 6b,d, Supplementary Figures 1b,d, 2b,d) but not when patch size variation was absent (green lines corresponding to memory of 1 in Figure 6a,c, Supplementary Figures 1a,c, 2a,c).

When patch size variation was present, individuals gained new associates for two reasons. First, because patch sizes varied, individuals were sometimes assigned to a small patch in an earlier time step but to a large patch in a later time step. If this occurred, there would not be sufficient previous associates to fill the larger patch in the later time step, and individuals would have to associate with an unknown individual from the population even if the associations were otherwise based on familiarity, resulting in a gain of new associates. Second, when associations were not always constant due to patch size variation, over time, it could so happen that even if an association was based on familiarity, a previous associate joining the group could be a previous associate of only some and not all individuals in the group. These two causes would lead to a gain in new associates and, over time, increased network connectivity and density (Figures 3, 5). However, if patch size variation was absent and memory was 1, there were always sufficient previous associates to fill patches, and individuals continued to be with their associates from the first time step. Therefore, at the end of 400 time steps, the network did not have any more connections than those at the first time step. Since each individual's associations were fully limited to previous associates, the resulting network was poorly connected and had low density (Figures 3, 5).

In all the simulations, if there was a gain of new associates, the rate of gain was faster when the average patch size was 4 (Figure 6a,b, Supplementary Figures 1a,b, 2a,b) than when it was 2 (Figure 3c,d, Supplementary Figures 1c,d, 2c,d) because there was a greater chance for an association to be formed with an unknown individual due to the reasons mentioned above. With an increase in average patch size, the time to meet all the individuals in the population would decrease, whereas an increase in memory would increase the time to meet all the individuals in the population (Figure 6).

Overall, associations were stable through time only when every association was with previous associates, which occurred when memory was 1 and patch size variation was absent. Thus,

the presence of memory alone (and therefore a familiarity-based rule) did not result in stable associations as long as unfamiliar individuals were allowed to use the patch. The above trends in the stability of associations were similar for memory lengths of 1, 20, and 400 (comparison of Figure 3, Supplementary Figures 1, 2), and only the last has been shown in the main text here.

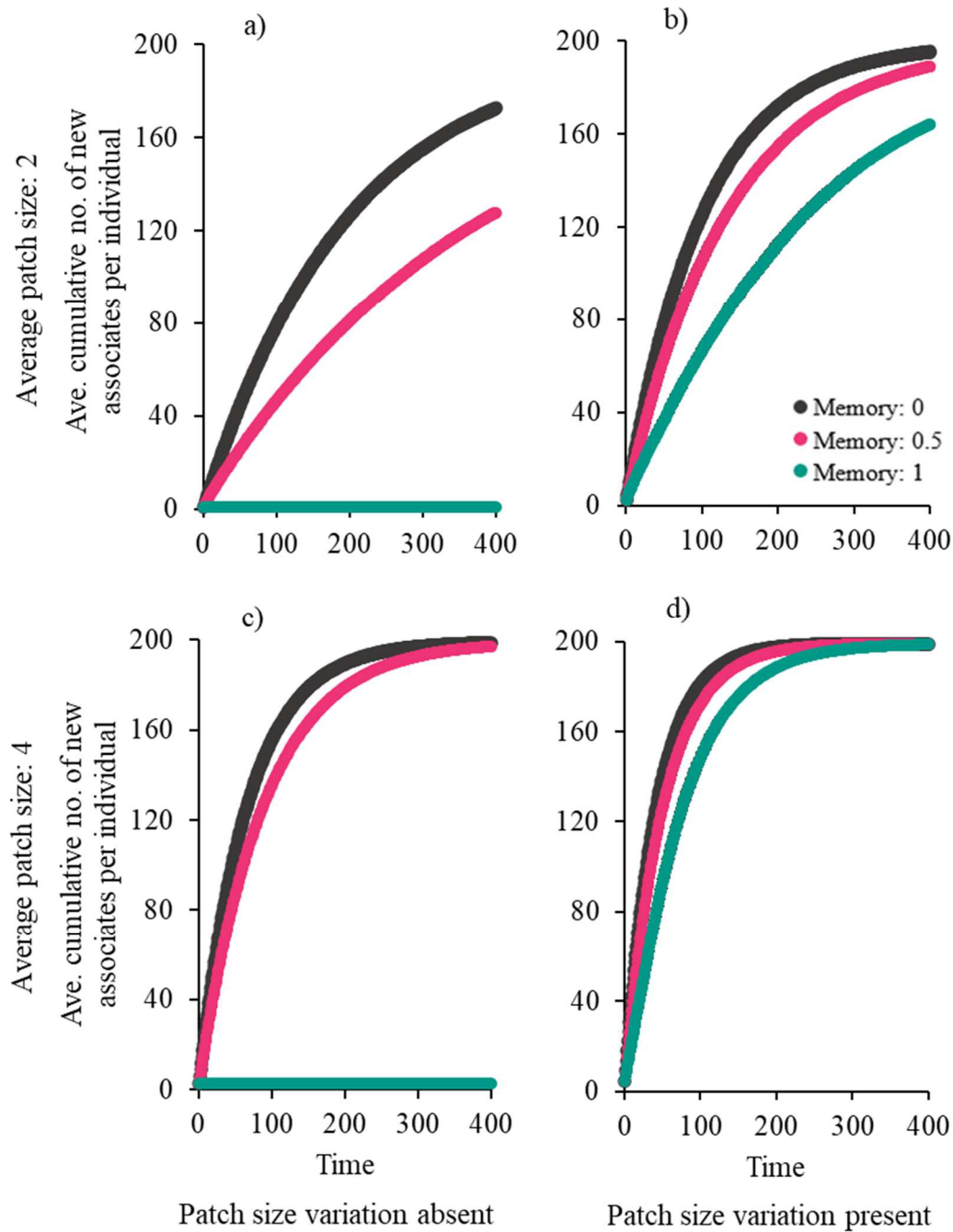


Figure 6. The plot of the average cumulative number of new associates per individual against time in simulations with a memory length of 400. The error bars are 95% CI about the mean but are too small to be clearly visible.

3.1.2 Network modularity

I found significant main effects of memory, memory length, average patch size, and patch size variation, as well as significant interaction effects of the two-way, three-way, and four-way interactions on the modularity of the network (Table 4). Similar to density, effect sizes were large only for memory ($\eta^2=0.439$), patch size variation ($\eta^2=0.198$), and the interaction effect between the two ($\eta^2=0.358$). The effect sizes for the main effects of average patch size and memory length, all the other interactions effects, and the error were close to zero (Table 4).

Table 4. Results of four factor ANOVA with network modularity as the dependent variable and memory length, patch size variation, and average patch size as the fixed independent variables. The large effect sizes are marked in bold.

Effect	<i>SS</i>	<i>df</i>	<i>MS</i>	<i>F</i>	<i>P</i>	η^2
{1}Memory	19.690	2	9.845	1375984.666	<0.001	0.439
{2}Memory length	0.023	2	0.011	1598.663	<0.001	0.001
{3}Average patch size	0.059	1	0.059	8269.221	<0.001	0.001
{4}Patch size variation	8.894	1	8.894	1243090.438	<0.001	0.198
Memory*Memory length	0.028	4	0.007	988.536	<0.001	0.001
Memory*Average patch size	0.006	2	0.003	439.236	<0.001	0.000
Memory length*Average patch size	0.009	2	0.004	622.469	<0.001	0.000
Memory*Patch size variation	16.094	2	8.047	1124711.087	<0.001	0.358
Memory length*Patch size variation	0.012	2	0.006	838.494	<0.001	0.000
Average patch size*Patch size variation	0.000	1	0.000	42.661	<0.001	0.000
Memory*Memory length*Average patch size	0.007	4	0.002	245.134	<0.001	0.000
Memory*Memory length*Patch size variation	0.038	4	0.010	1338.407	<0.001	0.001

Effect	<i>SS</i>	<i>df</i>	<i>MS</i>	<i>F</i>	<i>P</i>	η^2
Memory*Average patch size*Patch size variation	0.021	2	0.011	1480.587	<0.001	0.000
Memory length*Average patch size*Patch size variation	0.003	2	0.001	181.576	<0.001	0.000
1*2*3*4	0.012	4	0.003	420.260	<0.001	0.000
Error	0.002	324	0.000			0.000

Modularity was significantly higher (0.529) when associations were based on memory (memory=1) than when there was no role of memory (memory=0, modularity=0.026; Tukey's HSD test: $P<0.001$) (Figure 7a). Modularity at a memory of 0.5 (0.040) was similar to that at a memory of 0 (Figure 7a) but the difference was statistically significant ($P<0.001$). As mentioned above, although the main effect of memory length was significant, the effect size was negligible ($\eta^2 < 0.01$). Modularity at memory length of 400 was negligibly higher (0.209) than modularity at memory lengths of 20 and 1 (0.195, 0.191) but was statistically significant (Tukey's HSD tests: $P<0.001$; Figure 7b). Similarly, modularity was negligibly higher (0.211) when the average patch size was 2 than when the average patch size was 4 (0.185) (Figure 7c). Modularity was much higher when patch size variation was absent (0.355) than when it was present (0.041; Figure 7d).

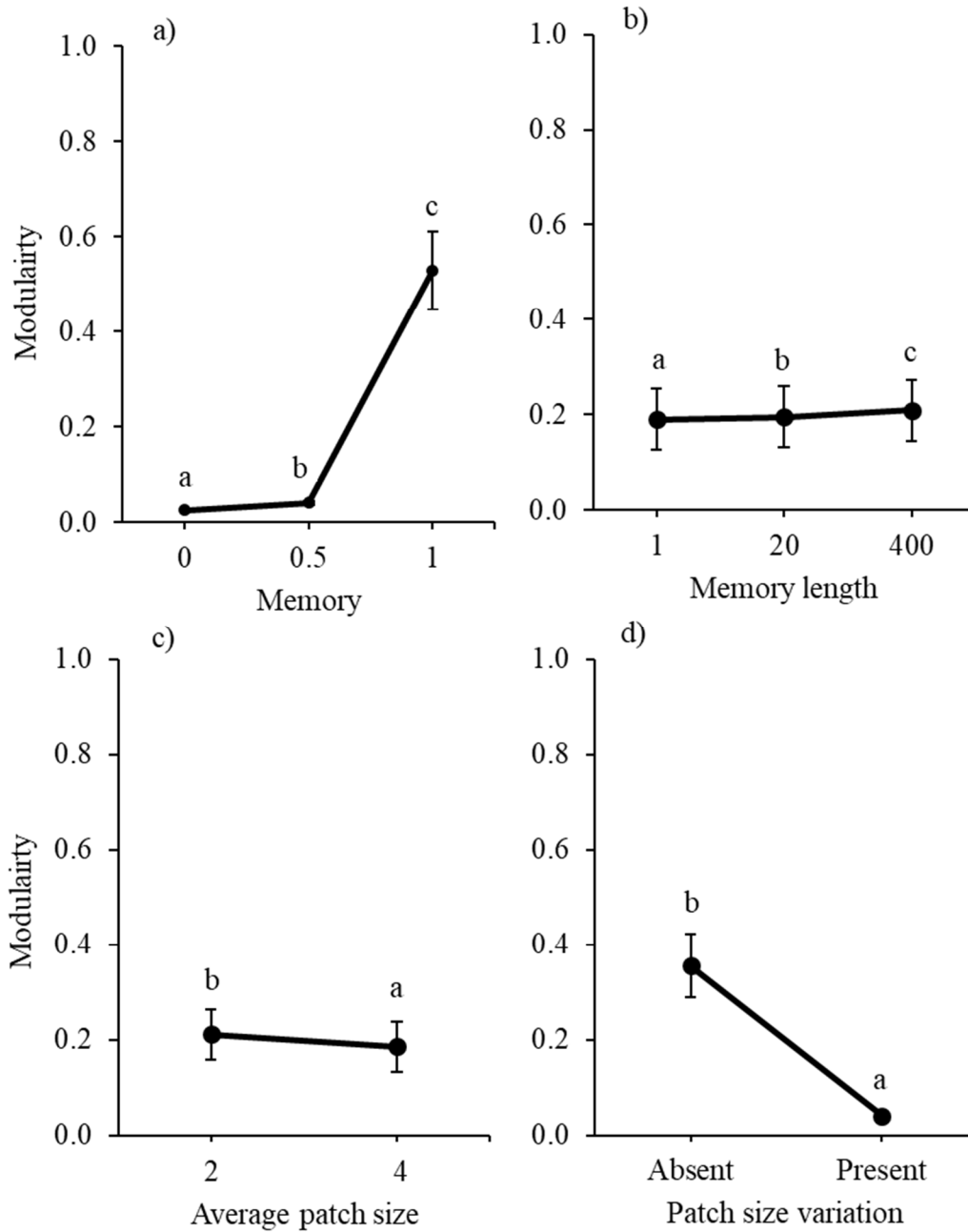


Figure 7. The main effect of memory, memory length, average patch size, and patch size variation on network modularity. Shared alphabets do not indicate significant difference, while different alphabets do, with $c > b > a$. The error bars are 95% CI about the mean.

Across different memory and patch size variation combinations, modularity was greater than 0.3, only when patch size variation was absent (0.985; Figure 8). This was because every association was with previous associates only in this case. When memory was 1 and patch size variation was present, individuals sometimes associated with unknown individuals either

to fill the patch whenever they could not find previous associates or the individuals added to the patch were previous associates of only one/some individuals in the patch. This resulted in lowered modularity (0.073; Tukey's HSD test: $P < 0.001$). When memory was 0 or 0.5, a lot of associations were random irrespective of patch size variation. For a given memory value, modularity was significantly different across simulations in which patch size variation was present (modularity=0.022 when memory=0, and 0.028 when memory=0.5) and those in which it was absent (modularity=0.030 when memory=0, and 0.052 when memory=0.5; Tukey's HSD tests: $P < 0.001$; Figure 8), even though the difference was marginal. The patterns seen in network modularity can also be visualised in terms of the CV of AI below.

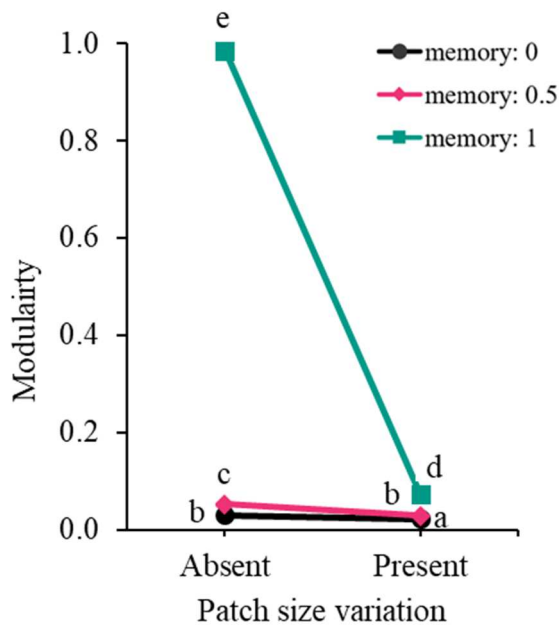


Figure 8. The interaction effect between memory and patch size variation on network modularity. When patch size variation was present, modularity was 0.022 at a memory of 0 and 0.028 at a memory of 0.5. Different alphabets indicate significant difference, with $e > d > c > b > a$. The error bars are 95% CI about the mean but are too small to be clearly visible.

3.1.2.1 Social differentiation: CV of association indices

Across all the simulations, the CV of AIs in the initial time steps was highest when memory was 1, moderately low when memory was 0.5, and lowest when memory was 0, indicating the difference in the extent of preferential associations occurring based on familiarity (Figure 9, Supplementary Figures 3, 4). Over time, irrespective of patch size variation, the CV of AIs

decreased when memory was 0 or 0.5 due to the formation of associations with random individuals in the population (black and pink lines corresponding to memory of 0 and 0.5, respectively, in Figure 9, Supplementary Figures 3, 4).

When memory was 1, CV of association indices remained high and the same throughout the 400-time steps only if patch size variation was absent (green line corresponding to memory of 0 and 0.5, respectively, in Figure 9a,c, Supplementary Figures 3a,c, 4a,c). This was because throughout the 400 time steps, individuals associated with their associates from the first time step (green lines corresponding to memory of 1 in Figure 6a,c, Supplementary Figures 1a,c, 2a,c). However, when memory was 1 and patch size variation was present, although CV of association indices was high, to begin with, it decreased over time due to the formation of connections with unknown individuals in the population either because there were no previous associates to associate with or because some individuals present in the group were previous associates of only one/some individuals in the group (green lines corresponding to memory of 1 in Figure 9b,d, Supplementary Figures 3b,d, 4b,d).

Thus, associations were strongly preferential and consistent through time only when every association was with previous associates, which was the case when memory was 1 and patch size variation was absent. These strong and consistent preferential associations resulted in distinct communities and a population with high modularity (Figure 8).

The above trends in the stability of associations were similar for memory lengths of 1, 20, and 400 (Figure 9, Supplementary Figures 3, 4), and only the last is shown in the main text.

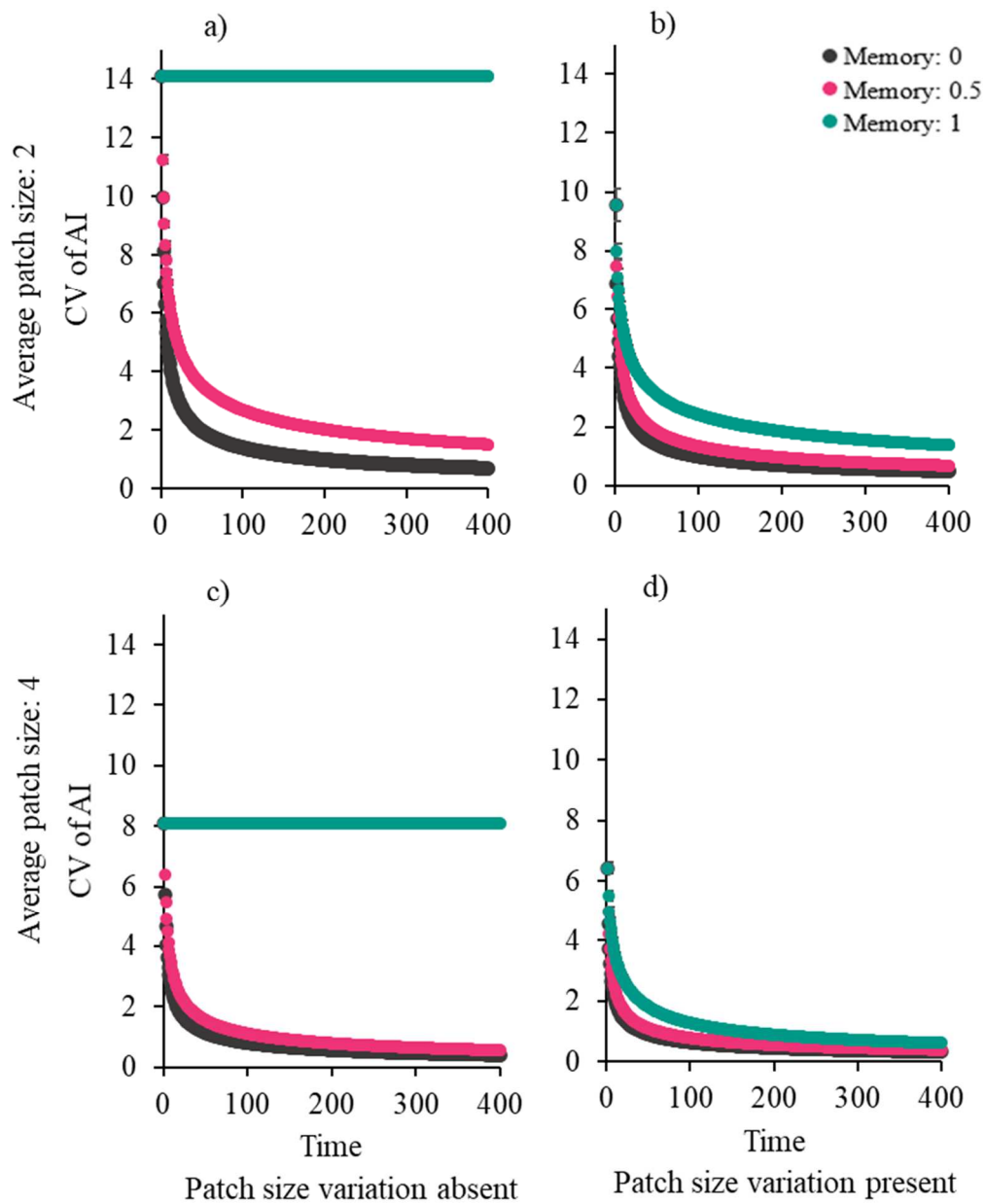


Figure 9. The plot of the CV of cumulative AI against time in simulations with a memory length of 400. The error bars are 95% CI about the mean but are too small to be clearly visible.

3.2 Under what conditions of inadequate sampling would social structure be wrongly inferred when individuals actually associate with one another randomly?

3.2.1 Network density

I found, from the three-factor ANOVA, significant main effects of sampling period length, sampling intensity, and average group size, and significant interaction effects of two-way and three-way interactions amongst the above three factors on the density of network (Table 5). The main effects of sampling period length ($\eta^2=0.661$) and sampling intensity ($\eta^2=0.212$) were large in terms of effect sizes, whereas average group size had a moderate effect size, and the interaction between sampling period length and sampling intensity, and the three-way interaction were small. The other effects were negligible (Table 5).

Table 5. Results of three-factor ANOVA with network density as the dependent variable and average group size, length of the sampling period, and sampling intensity as the fixed independent variables.

Effect	<i>SS</i>	<i>df</i>	<i>MS</i>	<i>F</i>	<i>P</i>	η^2
Average group size	2.337	1	2.337	45458.979	<0.001	0.071
Length of sampling period	21.654	2	10.827	210609.443	<0.001	0.661
Sampling intensity	6.938	4	1.735	33740.547	<0.001	0.212
Average group size*Length of sampling period	0.264	2	0.132	2565.032	<0.001	0.008
Average group size*Sampling intensity	0.022	4	0.005	105.702	<0.001	0.001
Length of sampling period*Sampling intensity	1.123	8	0.140	2731.306	<0.001	0.034
Average group size*Length of sampling period*Sampling intensity	0.432	8	0.054	1049.277	<0.001	0.013
Error	0.014	270	0.000			0.000

Density was highest when the length of the sampling period was 400 (0.778), moderate when sampling period length was 100 (0.415), and least when sampling period length was 20 (0.121; Tukey's HSD tests: $P < 0.001$; Figure 10a). Density of the network significantly increased with increase in sampling intensity from 0.2 (density=0.225) to 1 (density=0.682; Tukey's HSD tests: $P < 0.001$; Figure 10b). The main effect of average group size accounted for 7.1% of the variation in density, with density being higher when the average group size was 4 (0.527) than when the average group size was 2 (0.350) (Figure 10c). Amongst the interaction effects, sampling period length x sampling intensity had the largest effect size, with 3.4% of the variation in density being accounted for by this effect. For a given intensity of sampling, density was higher when the sampling period was longer, and for a given length of the sampling period, density was higher when sampling intensity was higher (Figure 11a,b). The increase in density with an increase in sampling intensity was steeper from a sampling intensity of 0.8 to 1 in the lower sampling periods than the sampling period of 400 (Figure 11a,b).

If we consider the density obtained at end of 400 time steps when 100% of the groups were sampled every time step as the true density (black squares in Figure 11a,b), for a given average group size, the density obtained through sampling was much closer to the true value when the sampling period was 400, and sampling intensity was 0.4 or higher (Figure 11a,b) because more associations were sampled (Figure 14f,i). Sample density was also close to the true value if the sampling intensity was very high (i.e., 100% of the groups were sampled every time step) and sampling period length was moderate (length=100) since greater number of associations were sampled (Figure 14h). Density obtained through sampling was closer to the true value when group sizes were larger (Figure 11b) than when group sizes were smaller (Figure 11a).

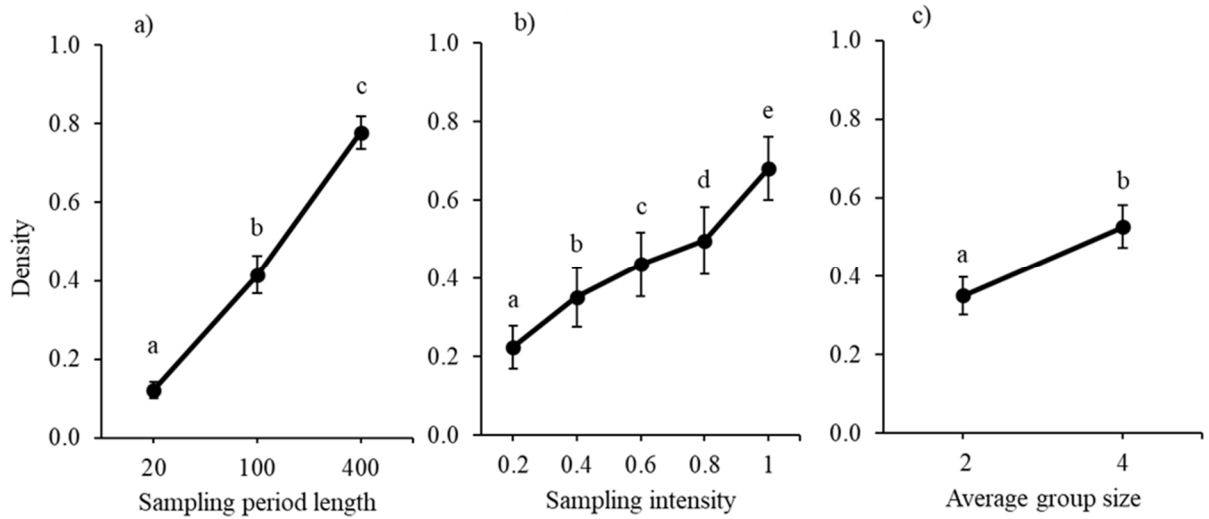


Figure 10. The main effect of sampling period length, sampling intensity, and average group size on network density. Different alphabets indicate significant difference, with $e > d > c > b > a$. The error bars are 95% CI about the mean.

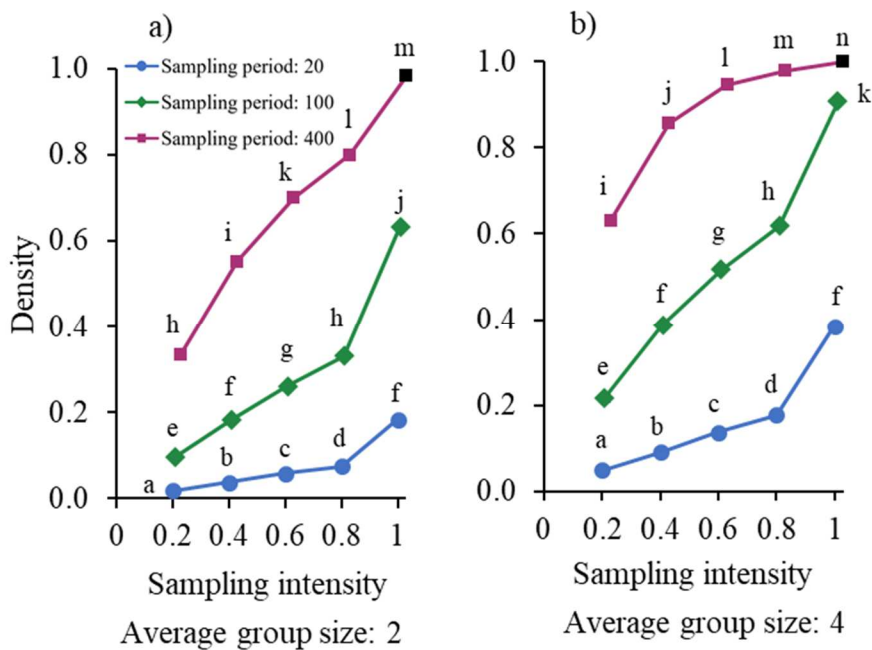


Figure 11. The effects of sampling period length, sampling intensity, and average group size on network density across different simulations. The black square corresponds to density in the simulation in which 100% of the groups were sampled every time step for 400 time steps. Different alphabets within panels indicate significant difference. The error bars are 95% CI about the mean but are too small to be visible.

3.2.2 Network modularity

I found significant main effects of sampling period length, sampling intensity, and average group size, as well as significant interaction effects of two-way and three-way interactions amongst the above three factors on network modularity based on the three-factor ANOVA (Table 6). The effect size was large ($\eta^2=0.607$) for the main effects of sampling period length, and sampling intensity ($\eta^2=0.212$), and moderate for the interaction effect between the two ($\eta^2=0.102$). The effect sizes for the main effect of average group size, other interaction effects, and the error were small (Table 6).

Table 6. Results of the three-factor ANOVA with network modularity as the dependent variable, and average group size, length of the sampling period, and sampling intensity as the fixed independent variables. The large effect sizes are marked in bold.

Effect	<i>SS</i>	<i>df</i>	<i>MS</i>	<i>F</i>	<i>P</i>	η^2
Average group size	0.354	1	0.354	2290.032	<0.001	0.046
Length of sampling period	4.692	2	2.346	15196.112	<0.001	0.607
Sampling intensity	1.640	4	0.410	2654.911	<0.001	0.212
Average group size*Length of sampling period	0.152	2	0.076	493.391	<0.001	0.020
Average group size*Sampling intensity	0.054	4	0.014	87.568	<0.001	0.007
Length of sampling period*Sampling intensity	0.785	8	0.098	635.958	<0.001	0.102
Average group size*Length of sampling period*Sampling intensity	0.017	8	0.002	14.124	<0.001	0.002
Error	0.042	270	0.000			0.005

Modularity was highest when the length of the sampling period length was 20 (0.342), lower when the sampling period was 100 (0.119), and least when the sampling period was 400 (0.048; Tukey's HSD: $P < 0.001$; Figure 12a). Modularity decreased with an increase in sampling intensity from 0.2 (modularity=0.289) to 1 (modularity=0.069; Tukey's HSD: $P < 0.001$; Figure 12b). The main effect of average group size accounted for 4.6% of the variation in modularity, with modularity being higher when the average group size was 2 (0.204) than when the average group size was 4 (0.135; Figure 12c). For a given sampling intensity, modularity was higher when the sampling period was shorter than when it was longer (Figure 13a,b). Further, for a given sampling period, modularity decreased when sampling intensity increased (Figure 13a,b). This decrease in modularity with sampling intensity was steeper for smaller sampling periods (Figure 13a,b).

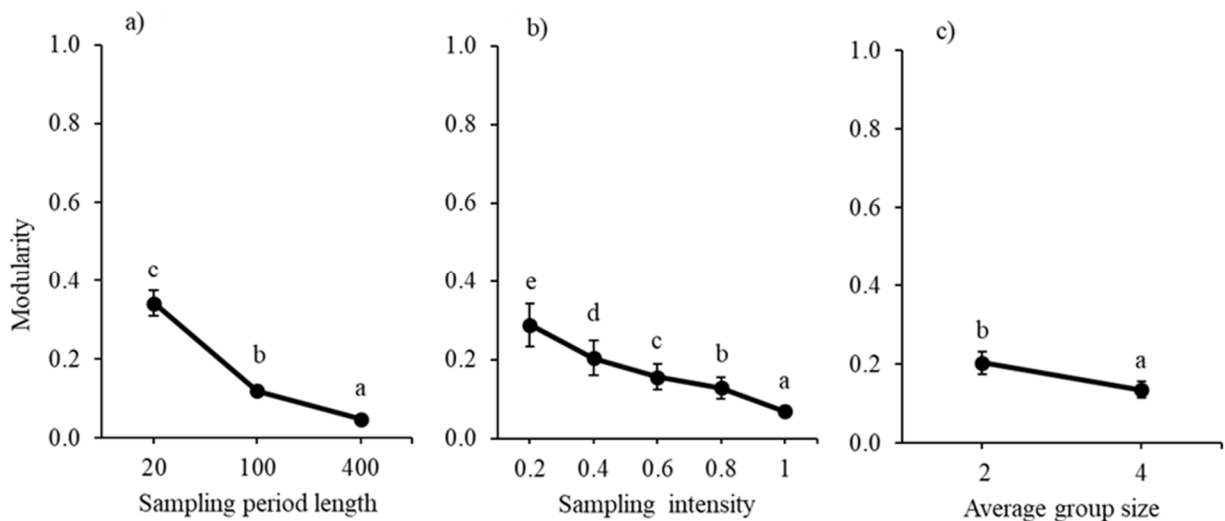


Figure 12. The main effect of sampling period length, sampling intensity, and average group size on network modularity. Different alphabets indicate significant difference, with $e > d > c > b > a$. The error bars are 95% CI about the mean.

Again, if we consider the modularity obtained at end of 400 time steps when 100% of the groups were sampled every time step as the true modularity (black squares in Figure 13), for a given average group size, the modularity obtained through sampling was much closer to the true value when the sampling period was long (length=400) irrespective of sampling intensity (Figures 13a,b) since a lot of the associations were sampled (Figure 14c,f,i). However, sample

modularity was also close to the true modularity if the sampling intensity was high (i.e., 80% to 100% of the groups were sampled every other time step and every time step, respectively) and sampling period length was moderate (length = 100). This was also because intense sampling resulted in sampling of many associations (Figure 14e,h). Similar to density, modularity obtained through sampling was slightly closer to the true value when the average group size was larger (4 versus 2; Figure 13b,a).

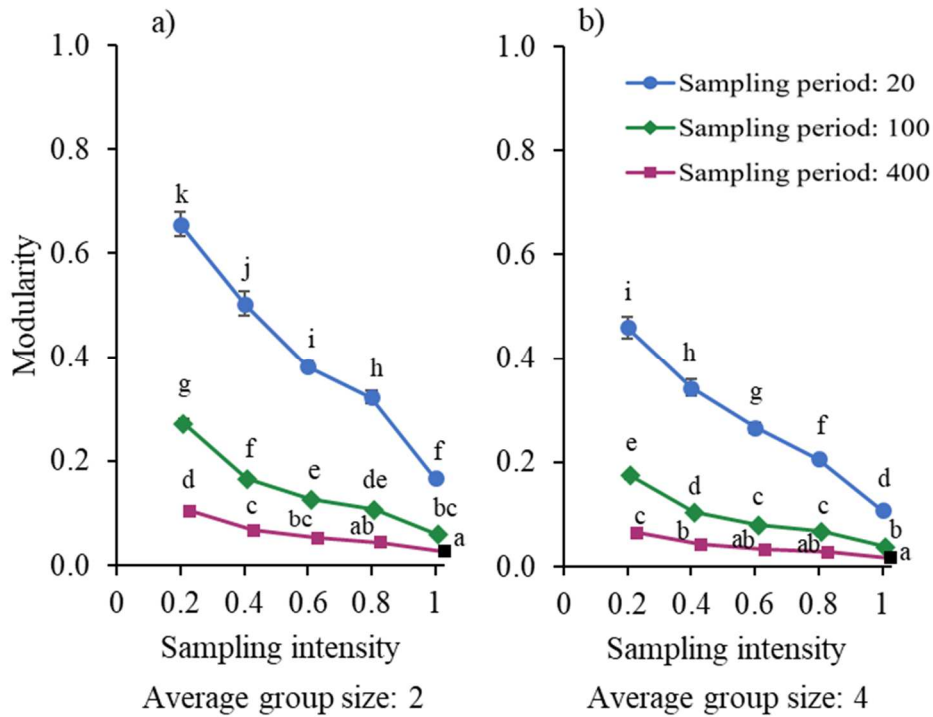


Figure 13. The effect of sampling period length, sampling intensity, and average group size on network modularity across different simulations. The black square corresponds to modularity in the simulation in which 100% of the groups were sampled every time step for 400 time steps. Different alphabets within panels indicate significant difference. The error bars are 95% CI about the mean but are too small to be visible clearly.

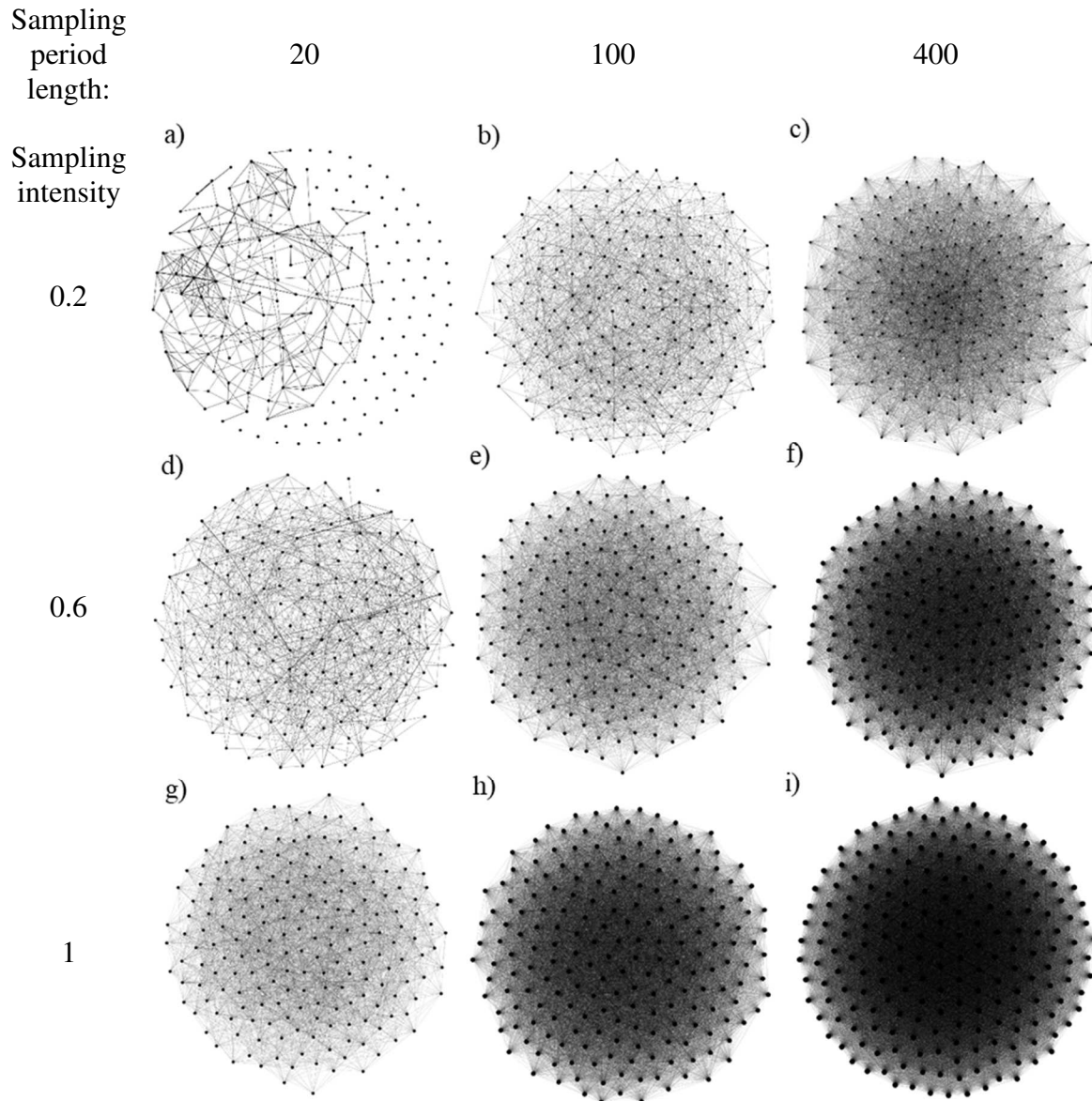


Figure 14. Association networks across three different sampling intensities and sampling periods when average group size was 2. Each network was created using sighting data from a randomly chosen sampling period of one of the 10 runs of each simulation.

CHAPTER 4: DISCUSSION

4.1 Does social structure emerge under different resource conditions when individuals associate based on familiarity?

Results from my simulations suggest that in a habitat with limited and patchy ephemeral resources, social structure characterised by low density, temporally stable associations, high modularity, and high social differentiation can occur only when every association is based on familiarity, which was the case when 100% of the associations were based on familiarity (memory parameter 1), and the habitat had the same amounts of resources regenerated every time step (patch size variation absent). However, a habitat without any variation in resource distribution is unlikely to occur in nature. I also found that over time, associations became similar to random associations when patchy ephemeral resources varied in quantity across patches and time steps (patch size variation present) even if 100% of the associations were to be based on familiarity (but could not be realised due to patch size variation and resource limitation). Overall, networks had lower density when the average group size was 2 than when the average group size was 4. The effect size of average group size on modularity was small, but this could be a result of taking average group size values that were not very different. In general, the above results indicate that a simple rule to associate based on familiarity is by itself not sufficient to explain social structure, and that resource conditions can strongly influence associations and social structure that emerges.

In my simulations, the rule to associate based on familiarity is similar to two other models (Rios and Kraenkel 2017, Cantor and Farine 2019). Rios and Kraenkel (2017) considered an agent-based model in which the agents moved in space depending on their memory of previous interactions. While moving randomly, if an agent encountered a neighbour, it carried out affiliative (move closer), agonistic (move away), or neutral (move randomly) interactions, depending on previous interactions. If the previous interaction was affiliative, the agent had a greater tendency to perform an affiliative interaction (move closer to the neighbour), and vice versa. At the end of 1000 time steps, multiple spatially separated groups with high modularity were obtained. Since the movement of animals is strongly affected by food resources, results from this model might not hold when resource conditions are also considered. Cantor and Farine (2018) built an agent-based model that considered a simple rule to forage with those that one last foraged with if the foraging resulted in the acquisition of sufficient resources. This model considered a habitat with a single resource patch that had

the same amount of resource at every time step. In the first time step, foraging ties were random, but after the first time step, ties were dependent on the outcome of competition between groups for the single resource patch. At a given time step, if a group obtained less resource (due to competition) than the number of individuals in the group, in the next time step, individuals within the group randomly removed one of the ties while if a group obtained more resources than the number of individuals in the group, a new tie with a random individual was created. However, when the resource share of a group was the same as the number of individuals, individuals were considered to obtain sufficient resources, and they did not change their ties, i.e., in the next time step, they continued to forage with previous associates. Over a short time, this rule resulted in the emergence of a single foraging group with stable group composition, provided the resource patch and the population size considered were small. Based on these results, the authors claimed that a simple rule of foraging could give rise to stable social groups and result in the emergence of foraging specialisation, especially in small populations with limited resources. Although not discussed by the authors, similar to my simulations, those results were very specific to the resource conditions considered: the presence of a single patch, small patch size, and absence of variation in resource quantity.

While the simulations discussed in the main text assumed resource limitation, I also separately examined scenarios in which this was not the case. As mentioned above, when patch size variation was present across time, since the resource was limited, some associations were sometimes forced with unknown individuals when previous associates were unavailable. If this assumption was relaxed such that there were many patches (many more feeding sites than individuals in the population), when memory parameter was 1, every association could occur with a familiar individual. I then found social structure characterised by low density and high modularity, irrespective of patch size variation (Supplementary Figure 5). However, such an abundance of resources may also be unrealistic. If resources were limited in certain time steps but abundant in others, as might be the case in nature, individuals might have some permanent companions and several casual acquaintances, resulting in a network with high density and high modularity. This is something that could be explored in the future.

The simulations I carried out also assumed that resources were ephemeral and had to be regenerated at every time step. If resources were not so ephemeral, individuals could visit the same patch during many time steps (or remain in the same patch) before moving on to another

patch; the social structure thus obtained could likely have low density and high modularity. Thus, the duration of unchanging resource patches relative to the interval at which individuals sort themselves or form new groupings could have an effect on social structure. Mynas leaving a roost every morning and arriving at different patches with insects that may be spatiotemporally unpredictable might correspond to the scenario I have modelled with patch size variation across space and time. (We do not know if there is social structure in mynas.)

While the simulations discussed here included a scenario with unchanging patches of equal sizes (which led to social structure being seen when 100% of the associations were based on familiarity), I also separately looked at unequal patch sizes that did not change over time. Food trees or sleeping sites for primates, or nesting sites for birds might be resources of this type. When patch sizes were unequal, and there was no variation in patch size across time, I did not find social structure (not shown) even if associations were to be based completely on familiarity because individuals in my model did not have a memory of the patch; they only had a memory or not of other individuals. Therefore, individuals from a larger patch in one time step could be split across smaller patches in the subsequent time step depending on the patch to which the first individual of a group was assigned. It would be worthwhile examining whether social structure can emerge with temporally and/or spatially varying patch sizes when individuals have a memory of the patch rather than of other individuals.

4.2 Question 2: Under what conditions of inadequate sampling would social structure be wrongly inferred when individuals actually associate with one another randomly?

Results from my simulations showed that a population can wrongly be inferred to be socially structured with modular communities and low density if sampling intensity is low and the sampling period is short. Using modularity as the metric, any intensity of sampling performed over long periods (400 time steps) reflected the true social structure better than high intensity sampling (sampling 100% of groups every day) over short periods (20 time steps long). Sample modularity also reflected the true value when a high intensity of sampling (80% or 100% of groups sampled) was conducted for moderately long periods of time. In the case of density, sample density was close to the true density when sampling intensity was moderate to intense (40% of more groups sampled) and sampling period was long (400 time steps), or if sampling intensity was high (100% of groups sampled) and sampling period was moderately long (100 time steps). In practice, sampling almost every group found at a given time in the field is close to impossible but sampling 40% or more groups every time step or

every other time step for long periods of time may be more feasible. Thus, to avoid wrong inferences of social structure, conducting long-term field studies with moderate intensity of sampling seems necessary.

Most studies that examine the social structure of a population collect data on associations and calculate different metrics. To detect the presence of preferential associations, standard deviation or CV of AI calculated for observed data is usually compared with that obtained for a null model (Whitehead *et al.* 2005). The null model, however, is created by permuting the observed data. Similarly, if modularity is used to detect community structure, the observed number of edges within communities is compared to that expected if observed associations were random. If associations observed do not reflect reality, as is the case when sampling is biased, social structure attributes calculated based on the associations would also not reflect reality. Statistical techniques to assess the precision and power of AI and metrics calculated using AI have been developed recently (Whitehead 2008b, Farine and Strandburg-Peshkin 2015, Shizuka and Farine 2016) but are not often used. Thus, it becomes pertinent to collect large amounts of data for long periods to infer social structure as close to reality as possible, as shown by our simulations and some others (Whitehead 2008b, Franks *et al.* 2010, Voelkl *et al.* 2011, Shizuka and Farine 2016, Murphy *et al.* 2021), especially when populations are poorly socially differentiated.

REFERENCES

1. Altizer S, Nunn CL, Thrall PH, Gittleman JL, Antonovics J, Cunningham AA, Dobson AP, Ezenwa V, Jones KE, Pedersen AB and Poss M. (2003). Social organisation and parasite risk in mammals: integrating theory and empirical studies. *Annual Review of Ecology, Evolution, and Systematics*, 34(1), 517–547.
2. Blondel VD, Guillaume JL, Lambiotte R and Lefebvre E. (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10), P10008.
3. Bodine EN, Panoff RM, Voit EO and Weisstein AE. (2020). Agent-based modeling and simulation in mathematics and biology education. *Bulletin of Mathematical Biology*, 82(8), 1–19.
4. Cairns SJ and Schwager SJ. (1987). A comparison of association indices. *Animal Behaviour*, 35(5), 1454–1469.
5. Cantor M and Farine DR. (2018). Simple foraging rules in competitive environments can generate socially structured populations. *Ecology and Evolution*, 8(10), 4978–4991.
6. Caraco T. (1980). Stochastic dynamics of avian foraging flocks. *The American Naturalist*, 115(2), 262–275.
7. Clutton-Brock TH. (1989). Review lecture: mammalian mating systems. *Proceedings of the Royal Society of London. B. Biological Sciences*, 236(1285), 339–372.
8. Clutton-Brock T and Janson C. (2012). Primate socioecology at the crossroads: past, present, and future. *Evolutionary Anthropology*, 21(4), 136–150.
9. Clutton-Brock TH and Huchard E. (2013). Social competition and selection in males and females. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 368(1631), 20130074.
10. Cohen JE. (1972). Markov population processes as models of primate social and population dynamics. *Theoretical Population Biology*, 3(2), 119–134.
11. Dehn MM. (1990). Vigilance for predators: detection and dilution effects. *Behavioural Ecology and Sociobiology*, 26(5), 337–342.
12. Farine DR and Strandburg-Peshkin A. (2015). Estimating uncertainty and reliability of social network data using Bayesian inference. *Royal Society Open Science*, 2(9), 150367.
13. Firth JA, Sheldon BC and Brent LJ. (2017). Indirectly connected: simple social differences can explain the causes and apparent consequences of complex social network positions. *Proceedings of the Royal Society B: Biological Sciences*, 284(1867), 20171939.

-
14. Franks DW, Ruxton GD and James R. (2010). Sampling animal association networks with the gambit of the group. *Behavioral Ecology and Sociobiology*, 64(3), 493–503.
 15. Gompper ME. (1996). Sociality and asociality in white-nosed coatis (*Nasua narica*): foraging costs and benefits. *Behavioral Ecology*, 7(3), 254–263.
 16. Hass CC and Valenzuela D. (2002). Anti-predator benefits of group living in white-nosed coatis (*Nasua narica*). *Behavioral Ecology and Sociobiology*, 51(6): 570–578.
 17. He P, Maldonado-Chaparro AA and Farine DR. (2019). The role of habitat configuration in shaping social structure: a gap in studies of animal social complexity. *Behavioral Ecology and Sociobiology*, 73(1), 1–14.
 18. Hinde RA. (1976). Interactions, relationships and social structure. *Man*, 11(1), 1–17.
 19. Ilany A and Akçay E. (2016). Social inheritance can explain the structure of animal social networks. *Nature Communications*, 7(1), 1–10.
 20. Janson CH and van Schaik CP. (1988). Recognising the many faces of primate food competition: methods. *Behaviour*, 105(1-2), 165–186.
 21. Kappeler PM and van Schaik CP. (2002). Evolution of primate social systems. *International Journal of Primatology*, 23(4), 707–740.
 22. Koenig A and Borries C. (2009). The lost dream of ecological determinism: time to say goodbye?... or a white queen's proposal? *Evolutionary Anthropology* 18(5), 166–174.
 23. Lusseau D and Newman ME. (2004). Identifying the role that animals play in their social networks. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 271, S477–S481.
 24. MATLAB. (2019). *Version 9.7.0.1471314 (R2019b)*. MathWorks Inc., Natick, Massachusetts.
 25. Murphy D, Wittemyer G, Henley MD and Mumby HS. (2021). Detecting community structure in wild populations: a simulation study based on male elephant, *Loxodonta africana*, data. *Animal Behaviour*, 174, 127–148.
 26. Nandini S, Keerthipriya P and Vidya TNC. (2018). Group size differences may mask underlying similarities in social structure: a comparison of female elephant societies. *Behavioral Ecology*, 29(1), 145–159.
 27. Newman ME. (2004). Analysis of weighted networks. *Physical Review E*, 70(5), 056131.
 28. Rios VP and Kraenkel RA. (2017). Do I know you? How individual recognition affects group formation and structure. *PLoS One*, 12(1), e0170737.
 29. Silk JB. (2007). The adaptive value of sociality in mammalian groups. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 362(1480), 539–559.
-

-
30. Sterck EH, Watts DP and Van Schaik CP. (1997). The evolution of female social relationships in nonhuman primates. *Behavioral Ecology and Sociobiology*, 41(5), 291–309.
 31. StatSoft, Inc. (2004). STATISTICA (data analysis software system), version 7. www.statsoft.com.
 32. Thierry B. (2008). Primate socioecology, the lost dream of ecological determinism. *Evolutionary Anthropology*, 17(2), 93–96.
 33. van Schaik CP. (1989). The ecology of social relationships amongst female primates. In Standen V and Foley RA (eds) *Comparative Socioecology*. Blackwell, Oxford, UK, pp.195–218.
 34. van Schaik CP and JARAM Van Hooff. (1983). On the ultimate causes of primate social systems. *Behaviour*, 85, 91–117.
 35. Voelkl B, Kasper C and Schwab C. (2011). Network measures for dyadic interactions: stability and reliability. *American Journal of Primatology*, 73(8), 731–740.
 36. Wasserman S and Faust K. (1994). *Social Network Analysis: Methods and Applications*. Cambridge University Press, Cambridge and New York, USA.
 37. Whitehead H. (2008a). *Analysing Animal Societies: Quantitative Methods for Vertebrate Social Analysis*. University of Chicago Press, Chicago, USA.
 38. Whitehead H. (2008b). Precision and power in the analysis of social structure using associations. *Animal Behaviour*, 75(3), 1093–1099.
 39. Wilson EO. (1975). *Sociobiology: The New Synthesis*. Belknap, Cambridge, USA.
 40. Wolf JB, Mawdsley D, Trillmich F and James R. (2007). Social structure in a colonial mammal: unravelling hidden structural layers and their foundations by network analysis. *Animal Behaviour*, 74(5), 1293–1302.
 41. Wrangham RW, Gittleman JL and Chapman CA. (1993). Constraints on group size in primates and carnivores: population density and day-range as assays of exploitation competition. *Behavioral Ecology and Sociobiology*, 32(3), 199–209.
 42. Wrangham RW. (1980). An ecological model of female-bonded primate groups. *Behaviour*, 75, 262–300

Supplementary Material 1

Table 1. List of simulations performed to examine the effect of memory, memory length, average patch size, and patch size variation on social structure

Simulation no.	Memory	Memory length	Average patch size	Patch size variation	Population size	Time	No. of replicates
1	1	1	2	Absent	200	400	10
2	1	1	2	Present	200	400	10
3	0.5	1	2	Absent	200	400	10
4	0.5	1	2	Present	200	400	10
5	0	1	2	Absent	200	400	10
6	0	1	2	Present	200	400	10
7	1	20	2	Absent	200	400	10
8	1	20	2	Present	200	400	10
9	0.5	20	2	Absent	200	400	10
10	0.5	20	2	Present	200	400	10
11	0	20	2	Absent	200	400	10
12	0	20	2	Present	200	400	10
13	1	400	2	Absent	200	400	10
14	1	400	2	Present	200	400	10
15	0.5	400	2	Absent	200	400	10
16	0.5	400	2	Present	200	400	10
17	0	400	2	Absent	200	400	10
18	0	400	2	Present	200	400	10
19	1	1	4	Absent	200	400	10
20	1	1	4	Present	200	400	10
21	0.5	1	4	Absent	200	400	10
22	0.5	1	4	Present	200	400	10
23	0	1	4	Absent	200	400	10
24	0	1	4	Present	200	400	10
25	1	20	4	Absent	200	400	10
26	1	20	4	Present	200	400	10

Simulation no.	Memory	Memory length	Average patch size	Patch size variation	Population size	Time	No. of replicates
27	0.5	20	4	Absent	200	400	10
28	0.5	20	4	Present	200	400	10
29	0	20	4	Absent	200	400	10
30	0	20	4	Present	200	400	10
31	1	400	4	Absent	200	400	10
32	1	400	4	Present	200	400	10
33	0.5	400	4	Absent	200	400	10
34	0.5	400	4	Present	200	400	10
35	0	400	4	Absent	200	400	10
36	0	400	4	Present	200	400	10

Table 2. List of simulations performed to examine the effect of sampling intensity, sampling period length and average group size on social structure.

Simulation no.	Number of periods	Sampling period length	Proportion of groups sampled	Sampling interval	Average group size	Population size	Total time	Number of replicates
1	20	20	0.2	2	2	200	400	10
2	20	20	0.4	2	2	200	400	10
3	20	20	0.6	2	2	200	400	10
4	20	20	0.8	2	2	200	400	10
5	20	20	1	1	2	200	400	10
6	4	100	0.2	2	2	200	400	10
7	4	100	0.4	2	2	200	400	10
8	4	100	0.6	2	2	200	400	10
9	4	100	0.8	2	2	200	400	10
10	4	100	1	1	2	200	400	10
11	1	400	0.2	2	2	200	400	10
12	1	400	0.4	2	2	200	400	10
13	1	400	0.6	2	2	200	400	10
14	1	400	0.8	2	2	200	400	10
15	1	400	1	1	2	200	400	10
16	20	20	0.2	2	4	200	400	10
17	20	20	0.4	2	4	200	400	10
18	20	20	0.6	2	4	200	400	10
19	20	20	0.8	2	4	200	400	10
20	20	20	1	1	4	200	400	10
21	4	100	0.2	2	4	200	400	10
22	4	100	0.4	2	4	200	400	10
23	4	100	0.6	2	4	200	400	10
24	4	100	0.8	2	4	200	400	10
25	4	100	1	1	4	200	400	10
26	1	400	0.2	2	2	200	400	10
27	1	400	0.4	2	2	200	400	10
28	1	400	0.6	2	2	200	400	10
29	1	400	0.8	2	2	200	400	10
30	1	400	1	1	2	200	400	10

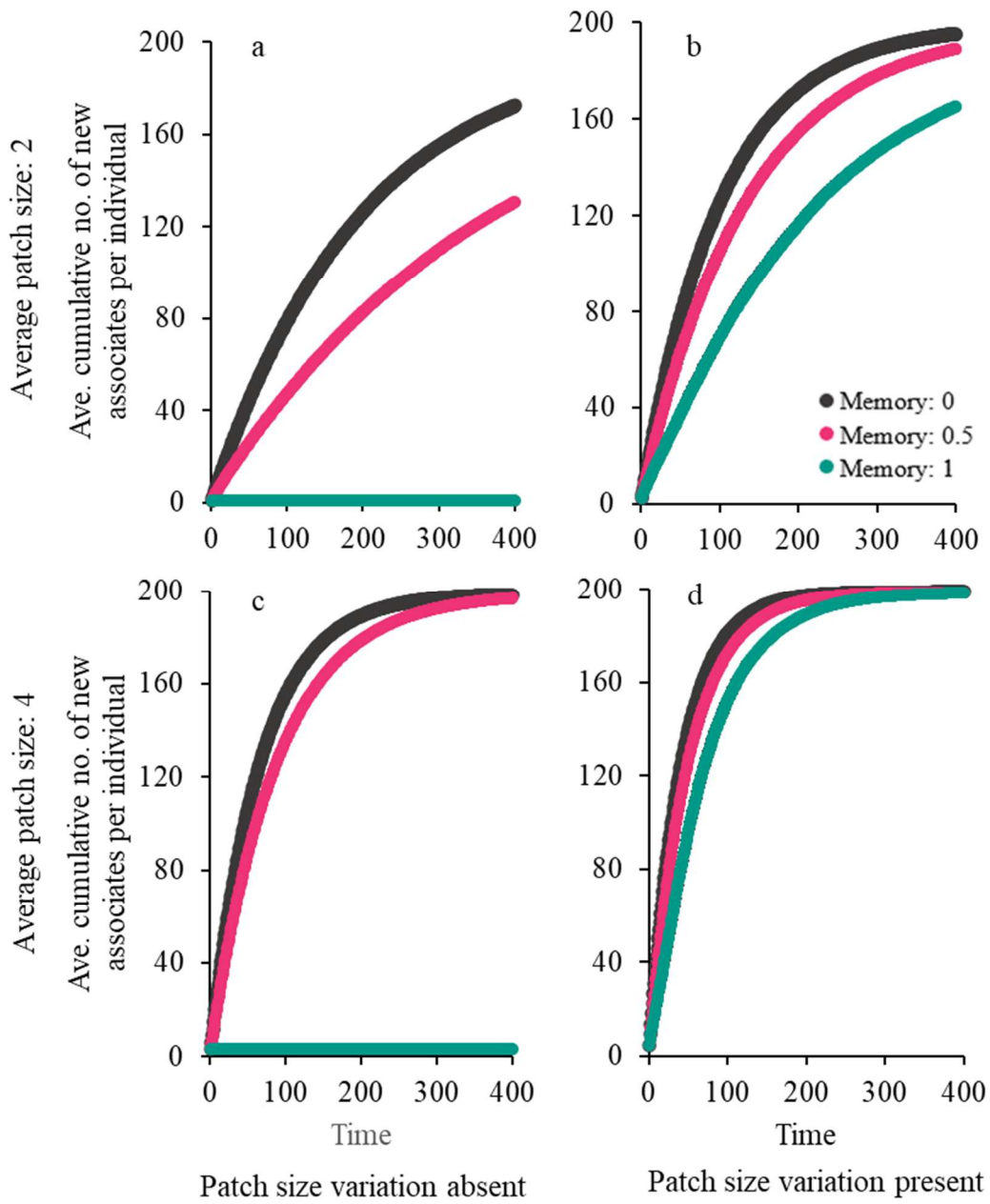


Figure 1. The plot of the average cumulative number of new associates per individual against time in simulations with a memory length of 20.

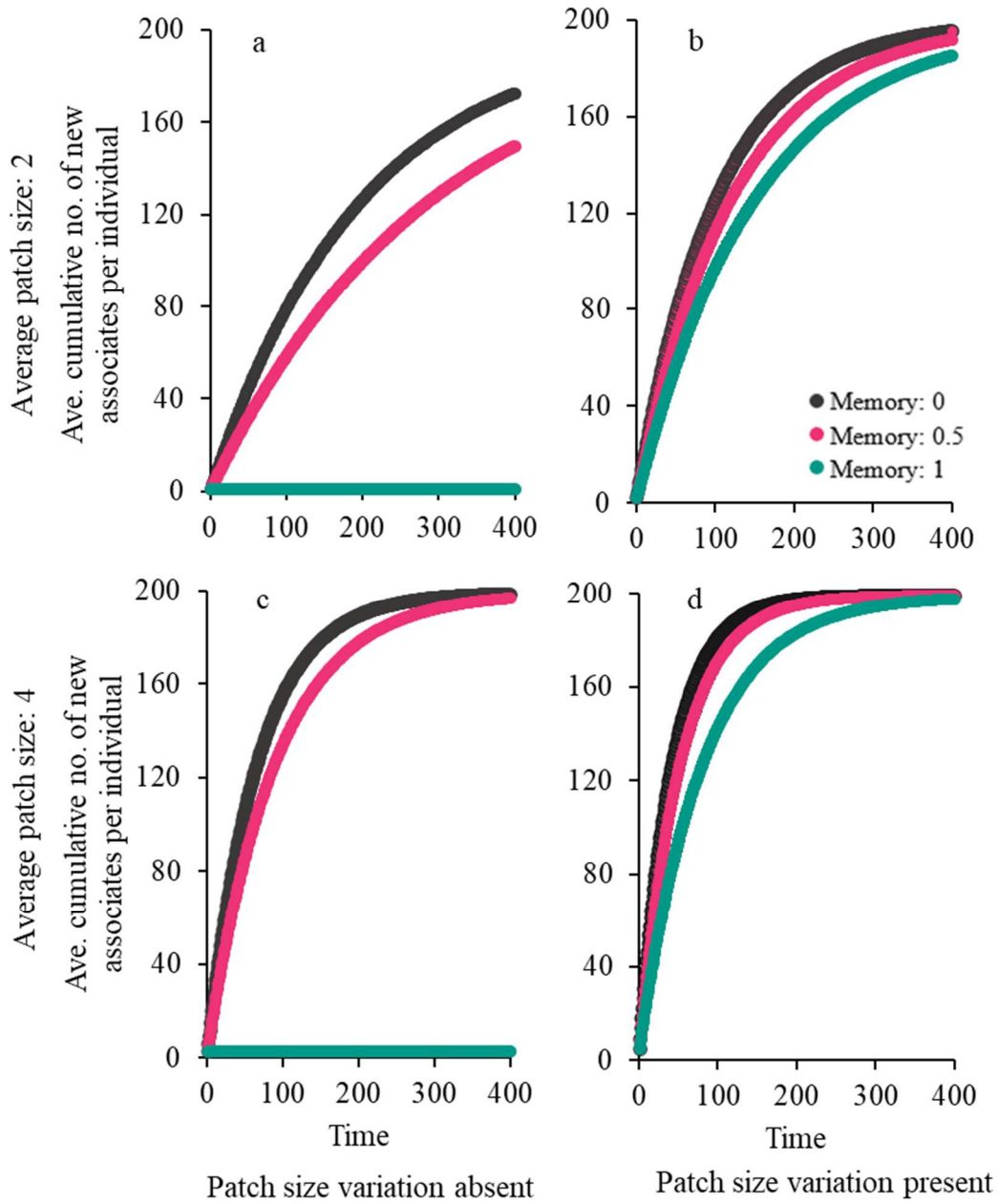


Figure 2. The plot of the average cumulative number of new associates per individual against time in simulations with a memory length of 1.

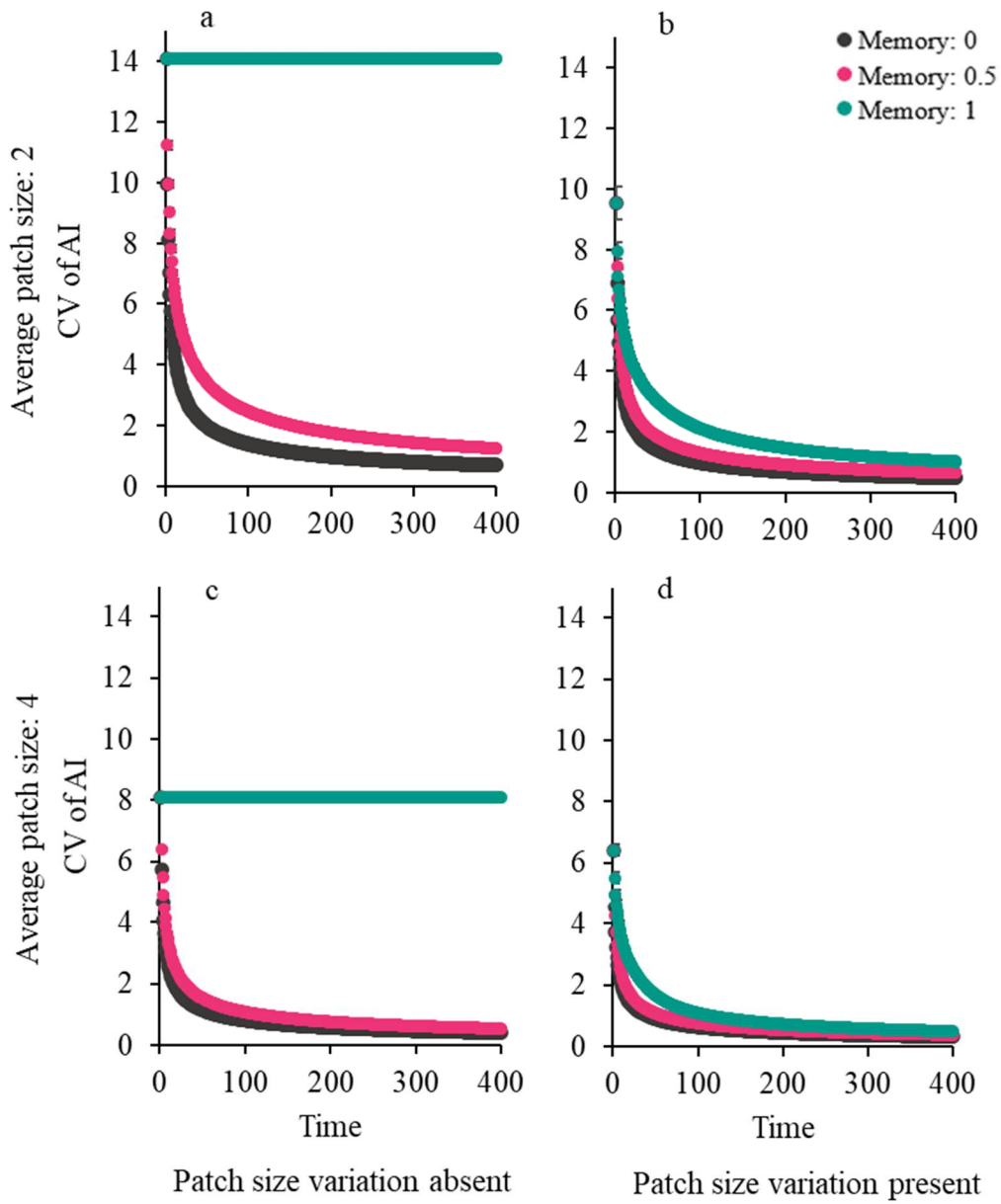


Figure 3. The plot of the CV of cumulative AI against time in simulations with a memory length of 20.

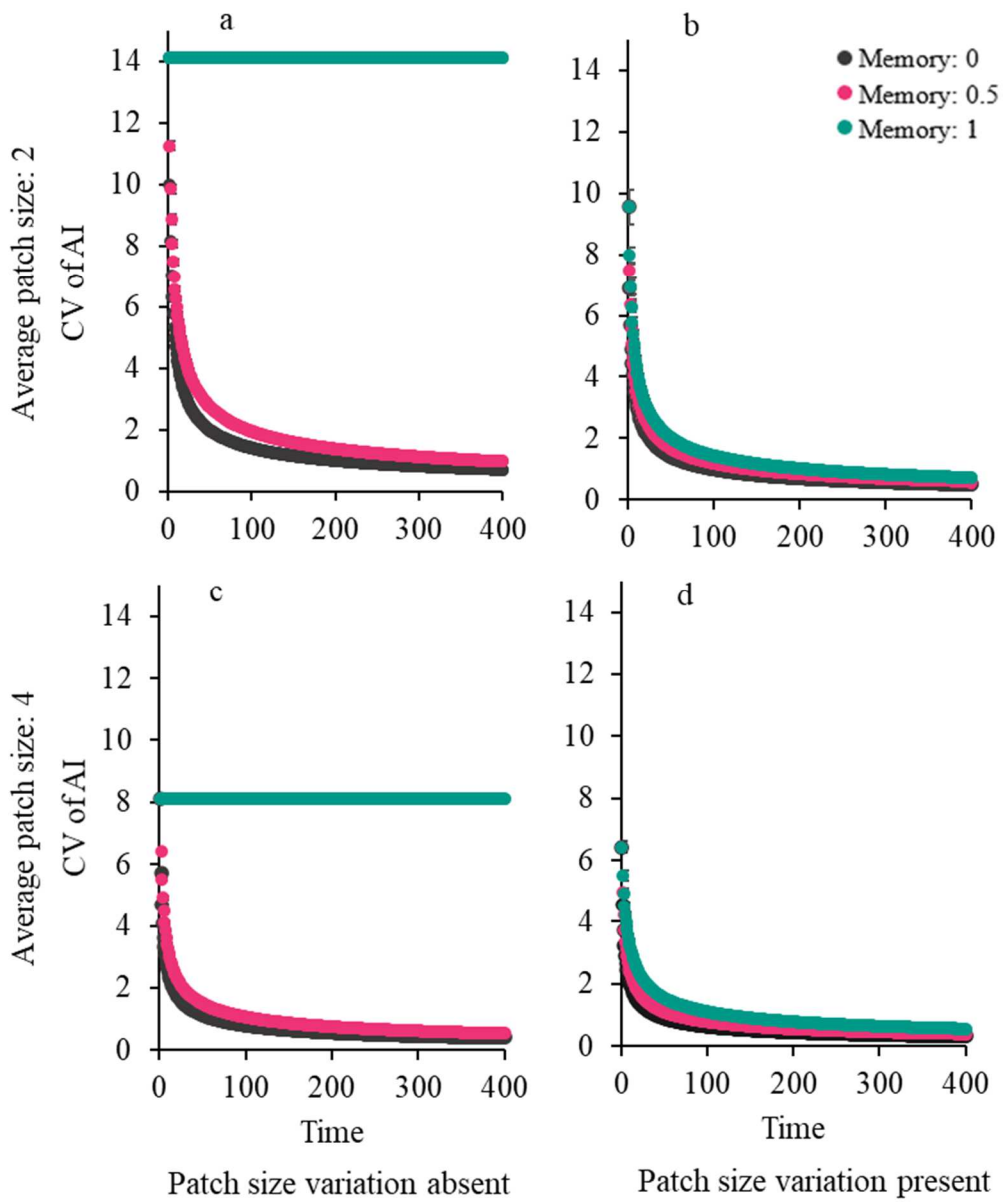


Figure 4. The plot of the CV of cumulative AI against time in simulations with a memory length of 1.

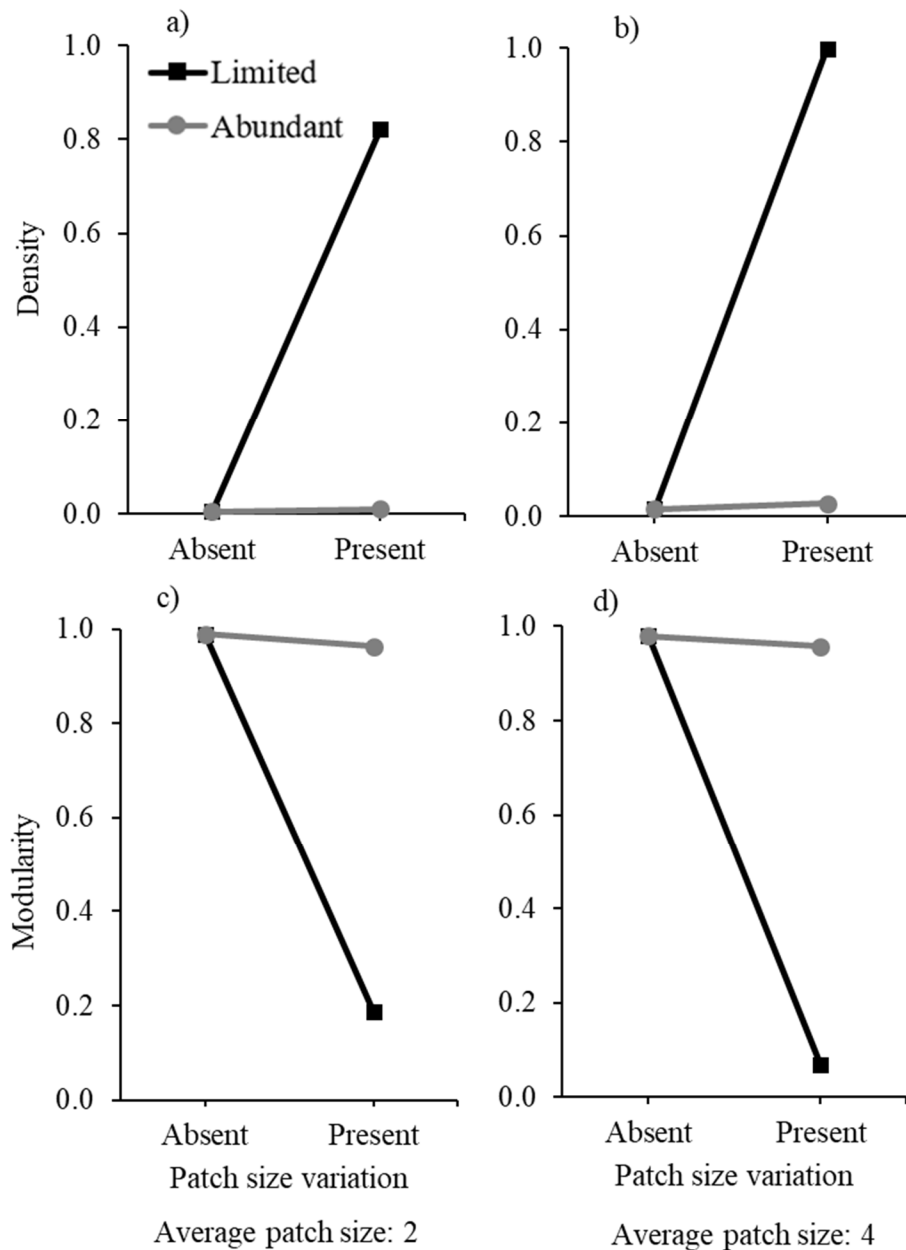


Figure 5. Plot of density and modularity in simulations in which resources were limited, and those in which resources were abundant when all the associations were based on familiarity (memory: 1), and memory length was 400. In simulations in which resources were abundant, the number of patches present was equal to the population size at every time step, i.e., there were enough patches to accommodate one individual in one patch, if necessary. In these simulations, individuals did not have to associate with unknown individuals if previous associates were unavailable.

Supplementary Material 2.

Memory_model_FL.m is the primary function that contains the codes to create patches, assign individuals to patches, generate sighting data, determine network statistics using sighting data, run the Louvain algorithm on the sighting data, and write the values of different variables into excel in simulations where risk is present and absent, respectively.

Memory_model_FL.m

For each data set replicate and every step, first, patch sizes are obtained by calling the Group_sizes.m function. This function generates patch sizes depending on two parameter values: gp_var_type and gp_size_type; gp_var_type can be 0 or 1, and gp_size_type can be 2 or 4. When gp_size type is 2, variables in the Group_sizes.m function are set such that the patch sizes, averaged across all time steps, is around 2. Similarly, when gp_size_type is 4, the patch sizes, averaged across all time steps, is around 4. When gp_var_type is 0, all the groups have equal group sizes, and the size remains the same across time. When gp_var_type is 1, every time step, patch sizes are drawn from a zero truncated negative binomial distribution. Once patch sizes are obtained, average, variance and median patch size for each time step is calculated.

Next, individuals are assigned to patches. In the first time step, individuals are randomly assigned to patches. For the remaining time steps, an individual is selected randomly from the population as the first individual of the first group. Then a uniformly random number between 0 and 1 is drawn. If this number is less than or equal to the memory parameter, the next individual of the patch is selected from the pool of previous associates of the first individual. The individuals to be included in the pool of previous associates is determined by the memory length parameter. If the random number drawn exceeds the memory parameter, the next individual is randomly selected from the population. When previous associates are selected, the probability of a previous associate being selected is weighted by the number of times it associated with the first individual. Based on the random number drawn, if the next individual to be assigned should be a previous associate, but if the first individual has not associated with anyone or does not remember associating with anyone, the next individual is randomly chosen from the population. Once an individual is assigned to a patch, it can not be assigned to another patch in the same time step. This ensures that individuals remain in the same patch

in a given time step. After the second individual is assigned, the process of drawing a random number and choosing the next individual is repeated until a given patch has as many individuals as the patch size. It is to be noted that when associations are with previous associates, and if more than one individual is already added to the patch, the probability of a new individual being selected is weighted by the number of times it associated, on average, with those individuals already present in the patch. The process of adding individuals to patches is repeated until all the patches have individuals assigned to them. Individuals within a patch are considered to be in one group and are associating with one another.

For a given time step, once individuals are assigned to patches, average, variance and median group size is calculated. Next, the group composition of all the groups is used to obtain ‘true’ sighting data for that time step (each group is one sighting). Further, group composition of 20% of the groups from a given time step, groups being chosen randomly, is used to obtain ‘sample’ sighting data for that time step. Sample sighting data, however, is obtained every two time steps. Finally, experienced group size of two kinds is calculated. In the first kind, average experienced group size = $\frac{\sum_{i=1}^n group\ size_i * group\ size_i}{\sum_{i=1}^n group\ size_i}$

In the second kind, the experienced group size of each individual is calculated for each time step; this is the group size of the individual minus 1.

Each time step, Adjacency_matrix.m function is used to obtain the adjacency matrix of associations in that time step. Adjacency matrix from first to the current time step is then used to obtain AI matrix. Further, mean, standard deviation and variance of AI matrix and mean, standard deviation and variance of adjacency matrix is calculated every time step.

The above process of obtaining patch sizes, calculating different patch/group size metrics, and assigning individuals to patches, and obtaining sighting data is repeated every time step.

After the last time step, the total true sighting data set is separated into periods that are 20 time steps long using Separate_sighting.m function. The same is repeated for the sample sighting data set. Further, variance in the average group size across time, variance in the experienced group size of type 2 across time, and average experienced group size of type 2 (averaged over different time steps for each individual) are calculated. The total sighting data (true and sample) is then modified to remove sighting data from the first twenty time steps.

This modified sighting data is given as input to `Network_metrics.m` function to obtain different network statistics. AI matrix obtained from `Network_metrics.m` is given as input to `Louvain_memory.m` function to run Louvain algorithm and obtain the number of levels/passes, the number of communities, maximum modularity reached in each pass (obtained through `Modularity.m` function), and community numbers assigned to each individual in different passes as output. The above process of obtaining network statistics and Louvain algorithm related statistics are repeated on the sighting data (true and sample) for each period that is 20-time steps long (excluding the first).

Creating patches, assigning individuals to patches, obtaining sighting data, generating network and Louvain statistics are repeated 10 times to get 10 different data replicates. Once there is data for all the different replicates, network statistics and Louvain statistics are written into excel using `Write_network_stats.m` and `Write_louvain_results.m` functions.

Further, for each replicate, using the AI matrix generated for every time step, the cumulative mean number of new associates is obtained using `New_associates.m` function. Finally, the following group/patch size summary statistics are calculated:

- 1) Patch size averaged across patches at each time step is averaged across time for different replicates;
- 2) Median patch size at each time step is averaged across time for different replicates;
- 3) Variance in patch size at each time step is averaged across time for different replicates;
- 4) Group size averaged across groups at each time step is averaged across time for different replicates;
- 5) Median group size at each time step is averaged across time for different replicates;
- 6) Variance in group size at each time step is averaged across time for different replicates;
- 7) Variance in average group size (averaged across groups) across time for different replicates;
- 8) The average experience group size of type 1 at each time step is averaged across time for different replicates;
- 9) The average experience group size of type 2 which is the group size experienced by each individual averaged across time, is averaged across all individuals for different replicates;
- 10) The variance in experience group size of type 2 which is the variance across time of the group size experienced by each individual, is averaged across all individuals for different replicates

Memory_model_FL.m

```
function [ticktock1, ticktock2] =
Memory_model_FL(simulation_no, pop_size, no_replicates, no_louvain_replicates, rng_val, ti
me, memory, memory_length, gp_size_type, gp_var_type, per_sample, sample_int, period_1,
period_2)

clc
initialVars = who('global');
clearvars('-except', initialVars{:})

mm = 0;
nn = 0;
pp = 0;

tic

true_sig = zeros(pop_size*time, pop_size);
sample_sig = zeros(pop_size*time, pop_size);
index = 1;
index2 = 1;
gp_sizes = zeros(1, pop_size);
assoc = zeros(pop_size, pop_size);
association = zeros(pop_size, pop_size);
true_no_gp_all_t = zeros(1, time);
sample_no_gp_all_t = zeros(1, time);
gp_sizes_all_t = zeros(time, pop_size);
gp_id_all_t = zeros(time, pop_size);
ave_gp_size = zeros(no_replicates, time);
med_gp_size = zeros(no_replicates, time);
ave_exp_gp_size = zeros(no_replicates, time);
var_gp_size = zeros(no_replicates, time);
exp_gp_size = zeros(time, pop_size);
var_ave_gp_size = zeros(no_replicates, 1);
var_exp_gp_size = zeros(no_replicates, pop_size);
ave_exp_gp_size_2 = zeros(no_replicates, pop_size);
cuml_adj = zeros(pop_size, pop_size);
cuml_ai_mean = zeros(no_replicates, time);
cuml_ai_var = zeros(no_replicates, time);
cuml_ai_std = zeros(no_replicates, time);
cuml_adj = zeros(pop_size, pop_size);
cuml_adj_mean = zeros(no_replicates, time);
cuml_adj_var = zeros(no_replicates, time);
cuml_adj_std = zeros(no_replicates, time);

true_freq_filter = 0;
true_ai = struct;
true_ave_sig_filt = struct;
```

```
true_no_inds_in_sig_data= struct;
true_no_AIs= struct;
true_ave_AI_filt= struct;
true_sd_AI_filt= struct;
true_skew_AI_filt= struct;
true_kurtosis_AI_filt= struct;
true_ave_deg= struct;
true_sd_deg= struct;
true_ave_weighted_deg= struct;
true_sd_weighted_deg= struct;
true_density= struct;
true_ave_CC= struct;
true_sd_CC= struct;
true_se_CC= struct;
true_CI_CC= struct;
true_ave_path_length= struct;
true_sd_path_length= struct;
true_se_path_length= struct;
true_CI_path_length= struct;
true_diameter= struct;
true_ave_eccentricity= struct;
true_sd_eccentricity= struct;
true_se_eccentricity= struct;
true_CI_eccentricity= struct;
true_top_ten_associates = struct;
true_top_five_associates = struct;
filt_true = struct;
```

```
true_ave_sig_filt_temp=zeros(no_replicates,1);
true_no_inds_in_sig_data_temp=zeros(no_replicates,1);
true_no_AIs_temp=zeros(no_replicates,1);
true_ave_AI_filt_temp=inf*ones(no_replicates,1);
true_sd_AI_filt_temp=inf*ones(no_replicates,1);
true_skew_AI_filt_temp=zeros(no_replicates,1);
true_kurtosis_AI_filt_temp=zeros(no_replicates,1);
true_ave_deg_temp=inf*ones(no_replicates,1);
true_sd_deg_temp=inf*ones(no_replicates,1);
true_ave_weighted_deg_temp=inf*ones(no_replicates,1);
true_sd_weighted_deg_temp=inf*ones(no_replicates,1);
true_density_temp=inf*ones(no_replicates,1);
true_ave_CC_temp=inf*ones(no_replicates,1);
true_sd_CC_temp=inf*ones(no_replicates,1);
true_se_CC_temp=inf*ones(no_replicates,1);
true_CI_CC_temp=inf*ones(no_replicates,1);
true_ave_path_length_temp=inf*ones(no_replicates,1);
true_sd_path_length_temp=inf*ones(no_replicates,1);
true_se_path_length_temp=inf*ones(no_replicates,1);
true_CI_path_length_temp=inf*ones(no_replicates,1);
true_diameter_temp=inf*ones(no_replicates,1);
true_ave_eccentricity_temp=inf*ones(no_replicates,1);
```

```
true_sd_eccentricity_temp=inf*ones(no_replicates,1);
true_se_eccentricity_temp=inf*ones(no_replicates,1);
true_CI_eccentricity_temp=inf*ones(no_replicates,1);
```

```
sample_freq_filter = 1;
sample_ai = struct;
sample_ave_sig_filt= struct;
sample_no_inds_in_sig_data= struct;
sample_no_AIs= struct;
sample_ave_AI_filt= struct;
sample_sd_AI_filt= struct;
sample_skew_AI_filt= struct;
sample_kurtosis_AI_filt= struct;
sample_ave_deg= struct;
sample_sd_deg= struct;
sample_ave_weighted_deg= struct;
sample_sd_weighted_deg= struct;
sample_density= struct;
sample_ave_CC= struct;
sample_sd_CC= struct;
sample_se_CC= struct;
sample_CI_CC= struct;
sample_ave_path_length= struct;
sample_sd_path_length= struct;
sample_se_path_length= struct;
sample_CI_path_length= struct;
sample_diameter= struct;
sample_ave_eccentricity= struct;
sample_sd_eccentricity= struct;
sample_se_eccentricity= struct;
sample_CI_eccentricity= struct;
filt_sample = struct;
sample_top_ten_associates = struct;
sample_top_five_associates = struct;
```

```
sample_ave_sig_filt_temp=zeros(no_replicates,1);
sample_no_inds_in_sig_data_temp=zeros(no_replicates,1);
sample_no_AIs_temp=zeros(no_replicates,1);
sample_ave_AI_filt_temp=inf*ones(no_replicates,1);
sample_sd_AI_filt_temp=inf*ones(no_replicates,1);
sample_skew_AI_filt_temp=zeros(no_replicates,1);
sample_kurtosis_AI_filt_temp=zeros(no_replicates,1);
sample_ave_deg_temp=inf*ones(no_replicates,1);
sample_sd_deg_temp=inf*ones(no_replicates,1);
sample_ave_weighted_deg_temp=inf*ones(no_replicates,1);
sample_sd_weighted_deg_temp=inf*ones(no_replicates,1);
sample_density_temp=inf*ones(no_replicates,1);
sample_ave_CC_temp=inf*ones(no_replicates,1);
```

```

sample_sd_CC_temp=inf*ones(no_replicates,1);
sample_se_CC_temp=inf*ones(no_replicates,1);
sample_CI_CC_temp=inf*ones(no_replicates,1);
sample_ave_path_length_temp=inf*ones(no_replicates,1);
sample_sd_path_length_temp=inf*ones(no_replicates,1);
sample_se_path_length_temp=inf*ones(no_replicates,1);
sample_CI_path_length_temp=inf*ones(no_replicates,1);
sample_diameter_temp=inf*ones(no_replicates,1);
sample_ave_eccentricity_temp=inf*ones(no_replicates,1);
sample_sd_eccentricity_temp=inf*ones(no_replicates,1);
sample_se_eccentricity_temp=inf*ones(no_replicates,1);
sample_CI_eccentricity_temp=inf*ones(no_replicates,1);
% filt_sample_temp = inf*ones(no_replicates,1);

true_sighting = struct;
sample_sighting = struct;
true_sig_all_rep = struct;
sample_sig_all_rep = struct;
true_no_gp_all_t_all_rep = struct;
sample_no_gp_all_t_all_rep = struct;
gp_id_all_t_all_rep = struct;
gp_sizes_all_t_all_rep = struct;

true_comm = struct;
true_modularity = struct;
true_num_levels = struct;
true_num_comm = struct;

sample_comm = struct;
sample_modularity = struct;
sample_num_levels = struct;
sample_num_comm = struct;

filename = strcat('Simulation_',num2str(simulation_no));
heading = {'Simulation no.' 'Population size' 'No. of replicates' 'No. of louvain replicates'
'No. of time steps', 'No. of large sampling periods','Memory' 'Memory length' 'Group size'
'Group size variation' 'Seed (for diff data set replicates)' };
xlswrite(filename,heading,'intial_conditions','a1');
xlswrite(filename,[simulation_no, pop_size, no_replicates,no_louvain_replicates,
time,period_1, memory, memory_length,
gp_size_type,gp_var_type],'intial_conditions','a2');
xlswrite(filename,rng_val,'intial_conditions','k2');
%

for rep = 1:no_replicates
rng(rng_val(rep))

for t = 1:time
disp(t)

```

```

%get patches/groups
gp_sizes = Group_sizes(gp_var_type,gp_size_type,pop_size,t,gp_sizes_all_t);

%patch/group size statistics
no_gp = sum(gp_sizes>0);
gp_sizes_all_t(t,:) = gp_sizes;
ave_gp_size(rep,t) = mean(gp_sizes(gp_sizes>0));
var_gp_size(rep,t) = var(gp_sizes(gp_sizes>0));
med_gp_size(rep,t) = median(gp_sizes(gp_sizes>0));

%assign individuals to groups/patches in the first time step
if t==1
    %create group id variable. It has group id for all inds. This will have identity of groups
    repeated based on group size
    gp_id = repelem(1:no_gp,gp_sizes(gp_sizes>0));
    gp_id = gp_id(randperm(length(gp_id))); %this randomises the group id of all inds.
    Now inds have random group ids i.e. are assigned to groups randomly.
elseif t>1
    inds = 1:pop_size; %individuals to choose from
    gp_id = zeros(1,pop_size);
    rowname = 1:pop_size; %row names of the assoc weights matrix
    colname = 1:pop_size; %col names of the assoc weights matrix
    for i = 1:no_gp %assign individuals to groups one at a time
        counter = 1; %counter for each individual in a given group
        sel_ind = zeros(1,pop_size); %vector of individuals selected
        sel_ind(counter) = randsample(repelem(inds(inds>0),2),1); %select the first ind of
the group randomly
        col_sel_ind = find(colname==sel_ind(counter)); %find the column of the selected
individual in the assoc weights matrix %
        assoc(:, col_sel_ind) = 0; %remove the column of the selected
individual in the assoc weights matrix
        colname(col_sel_ind) = 0; %column names after removing the column
of the selected individual in the assoc weights matrix
        while counter<gp_sizes(i) %to assign remaining individuals of a
group
            counter = counter+1;
            inds(ismember(inds,sel_ind)) = 0; %remove already chosen individuals
to prevent them from getting selected again
            gg = rand;
            if memory>=gg %check if associations are based on
familiarity or are random
                %if associations are based on familiarity
                row_sel_ind = ismember(rowname,sel_ind); %find the row number of the
selected ind(s)
                assoc_sel_ind = assoc(row_sel_ind,:); %get assoc weights from assoc
weights matrix for the sel ind(s)
                combined_assoc_sel_ind = mean(assoc_sel_ind,1); %get average assoc
weights of all selected individuals
                if mean(combined_assoc_sel_ind)~=0 %to check if selected inds have 0

```

assoc weights with all the remaining inds, i.e.,they haven't previously associated with any of the remaining individuals

```
    mm = mm+1;
    sel_ind(counter) =
randsample(repelem(inds,2),1,true,repelem(combined_assoc_sel_ind,2));
    else
        nn = nn+1;
        sel_ind(counter) = randsample(repelem(inds(inds>0),2),1); %if selected
inds have 0 assoc weights with all remaining inds, next individual is chosen uniformly
randomly
    end
    else
        pp = pp+1; %if associations are random
        sel_ind(counter) = randsample(repelem(inds(inds>0),2),1); %all individuals
whether known or unknown have equal chance of being selected
    end
        col_sel_ind = find(colname==sel_ind(counter)); %find the column of the
newly selected individual
        assoc(:, col_sel_ind) = 0; %remove the column of the newly selected
individual from the assoc matrix
        colname(col_sel_ind) = 0; %remove the column name of the newly selected
individual from the assoc matrix

    end
    gp_id(sel_ind(sel_ind>0)) = i; %provide group id for the sel ind which are
numbers greater than 0
    inds(ismember(inds,sel_ind)) = 0; %remove inds already selected

end
end
```

%calculate average experienced group size every t
dummy = 0;

```
for i = 1:no_gp
    dummy = dummy + gp_sizes(i)*gp_sizes(i);
end
ave_exp_gp_size(rep,t) = dummy/sum(gp_sizes);
```

%second kind of experienced group size

```
for i = 1:pop_size
    focal_ind_id = gp_id(i);
    exp_gp_size(t,i) = sum(gp_id==focal_ind_id)-1;
end
```

%get 100% of the sighting data by sampling. Based on group ids assigned to individuals
sighting data is obtained

```
for i = 1:no_gp
    temp = find(gp_id==i); %find all inds assigned to the same group
    true_sig(index,1:length(temp)) = temp; %each group is one sighting
    index = index+1;
```

```

end
true_no_gp_all_t(t) = no_gp;           %to keep track of the true number of groups in
each time step

%get a percentage of the sighting data by sampling every few time steps
if rem(t,sample_int)==0
    sample_no = ceil(per_sample*no_gp/100);    %number of groups that will be sampled in
this time step
    sample_gp = randsample(1:no_gp, sample_no); %to obtain ids of groups that will be
sampled in this time step through random draw
    for i = 1:sample_no
        temp = find(gp_id==sample_gp(i));      %to find individuals belonging to the group id
chosen randomly
        sample_sig(index2,1:length(temp)) = temp; %assign group to sighting data
        index2 = index2+1;
    end
    sample_no_gp_all_t(t) = sample_no;        %to keep track of the number of groups
sampled in every time step
end

%adjacency matrix is created every time step
field = strcat('t',num2str(t));
adj_every_t.(field) = Adjacency_matrix(gp_id,pop_size);
association = association+adj_every_t.(field);
assoc = zeros(pop_size, pop_size);
fieldname = strcat('period_2_',num2str(t));
true_ai(rep).(fieldname) = adj_every_t.(field);
cuml_adj = cuml_adj + adj_every_t.(field);
cuml_ai_mean(rep,t) = mean(cuml_adj/t,'all');
cuml_ai_var(rep,t) = var(cuml_adj/t,0,'all');
cuml_ai_std(rep,t) = std(cuml_adj/t,0,'all');
cuml_adj_mean(rep,t) = mean(cuml_adj,'all');
cuml_adj_var(rep,t) = var(cuml_adj,0,'all');
cuml_adj_std(rep,t) = std(cuml_adj,0,'all');

%association weights for the next time step
if t<=memory_length
    for i = 1:t
        field = strcat('t',num2str(i));
        assoc = assoc + adj_every_t.(field);    %depending on the memory length adjacency
matrices are summed to get association weights
    end
else
    for i = t-memory_length+1:t
        field = strcat('t',num2str(i));
        assoc = assoc + adj_every_t.(field);
    end
end

```

```

gp_id_all_t(t,:) = gp_id;           %to keep track of group ids of all individuals every
time step
gp_id = 0;
gp_sizes = zeros(1,pop_size);

end   %end of time loop

%variance in group size across t, variance in a kind of exp
var_ave_gp_size(rep) = var(ave_gp_size(rep,:));
var_exp_gp_size(rep,:) = var(exp_gp_size);
ave_exp_gp_size_2(rep,:) = mean(exp_gp_size);

%to separate sighting data based on periods and assigning data from each period into a
struct
true_sighting(rep).r = Separate_sighting(true_no_gp_all_t,true_sig, time, period_1,
period_2);
sample_sighting(rep).r = Separate_sighting(sample_no_gp_all_t,sample_sig, time, period_1,
period_2);

%get overall AI for all time steps leaving out first 20 time steps
fieldname = strcat('period_1_',num2str(0));
dummy = sum(true_no_gp_all_t(1:20));
temp_true_sig = true_sig(dummy+1:end, :);
dummy = sum(sample_no_gp_all_t(1:20));
temp_sample_sig = sample_sig(dummy+1:end, :);
[true_ai(rep).(fieldname),~,
true_no_inds_in_sig_data(rep).(fieldname),true_ave_sig_filt(rep).(fieldname),true_no_AIs(r
ep).(fieldname),true_ave_AI_filt(rep).(fieldname),true_sd_AI_filt(rep).(fieldname),true_ske
w_AI_filt(rep).(fieldname),true_kurtosis_AI_filt(rep).(fieldname),true_ave_deg(rep).(fieldn
ame),true_sd_deg(rep).(fieldname),true_ave_weighted_deg(rep).(fieldname),true_sd_weight
ed_deg(rep).(fieldname),true_density(rep).(fieldname),true_ave_CC(rep).(fieldname),true_s
d_CC(rep).(fieldname),true_se_CC(rep).(fieldname),true_CI_CC(rep).(fieldname),true_ave
_path_length(rep).(fieldname),true_sd_path_length(rep).(fieldname),true_se_path_length(re
p).(fieldname),true_CI_path_length(rep).(fieldname),true_diameter(rep).(fieldname),true_av
e_eccentricity(rep).(fieldname),true_sd_eccentricity(rep).(fieldname),true_se_eccentricity(r
ep).(fieldname),true_CI_eccentricity(rep).(fieldname)] =
Network_metrics(temp_true_sig,true_freq_filter,pop_size,0,2.262);
[true_louvain_ind, true_comm(rep).(fieldname), true_modularity(rep).(fieldname),
true_num_comm(rep).(fieldname), true_num_levels(rep).(fieldname)] =
Louvain_memory(true_ai(rep).(fieldname),rep,0,'period_1',0,simulation_no,no_louvain_rep
licates); %0 for true data
filt_true(rep).(fieldname) = length(true_louvain_ind);
[sample_ai(rep).(fieldname),~,
sample_no_inds_in_sig_data(rep).(fieldname),sample_ave_sig_filt(rep).(fieldname),sample
_no_AIs(rep).(fieldname),sample_ave_AI_filt(rep).(fieldname),sample_sd_AI_filt(rep).(fiel
dname),sample_skew_AI_filt(rep).(fieldname),sample_kurtosis_AI_filt(rep).(fieldname),sa
mple_ave_deg(rep).(fieldname),sample_sd_deg(rep).(fieldname),sample_ave_weighted_deg
(rep).(fieldname),sample_sd_weighted_deg(rep).(fieldname),sample_density(rep).(fieldnam
e),sample_ave_CC(rep).(fieldname),sample_sd_CC(rep).(fieldname),sample_se_CC(rep).(fi

```

```

eldname),sample_CI_CC(rep).(fieldname),sample_ave_path_length(rep).(fieldname),sample_
e_sd_path_length(rep).(fieldname),sample_se_path_length(rep).(fieldname),sample_CI_pat
h_length(rep).(fieldname),sample_diameter(rep).(fieldname),sample_ave_eccentricity(rep).(
fieldname),sample_sd_eccentricity(rep).(fieldname),sample_se_eccentricity(rep).(fieldname
),sample_CI_eccentricity(rep).(fieldname)] =
Network_metrics(temp_sample_sig,sample_freq_filter,pop_size,1,2.262);
[sample_louvain_ind, sample_comm(rep).(fieldname), sample_modularity(rep).(fieldname),
sample_num_comm(rep).(fieldname), sample_num_levels(rep).(fieldname)] =
Louvain_memory(sample_ai(rep).(fieldname),rep,1,'period_1',0,simulation_no,no_louvain_
replicates); %0 for sample data
filt_sample(rep).(fieldname) = length(sample_louvain_ind);
temp_true_sig = [];
temp_sample_sig = [];

```

```

%%get AI seperately for each sampling period except the first

```

```

for i = 2:period_1

```

```

    fieldname = strcat('period_1_',num2str(i));
    temp_true_sig = true_sighting(rep).r.(fieldname);
    temp_sample_sig = sample_sighting(rep).r.(fieldname);
    [true_ai(rep).(fieldname), ~,
true_no_inds_in_sig_data(rep).(fieldname),true_ave_sig_filt(rep).(fieldname),true_no_AIs(r
ep).(fieldname),true_ave_AI_filt(rep).(fieldname),true_sd_AI_filt(rep).(fieldname),true_ske
w_AI_filt(rep).(fieldname),true_kurtosis_AI_filt(rep).(fieldname),true_ave_deg(rep).(fieldn
ame),true_sd_deg(rep).(fieldname),true_ave_weighted_deg(rep).(fieldname),true_sd_weight
ed_deg(rep).(fieldname),true_density(rep).(fieldname),true_ave_CC(rep).(fieldname),true_s
d_CC(rep).(fieldname),true_se_CC(rep).(fieldname),true_CI_CC(rep).(fieldname),true_ave
_path_length(rep).(fieldname),true_sd_path_length(rep).(fieldname),true_se_path_length(re
p).(fieldname),true_CI_path_length(rep).(fieldname),true_diameter(rep).(fieldname),true_av
e_eccentricity(rep).(fieldname),true_sd_eccentricity(rep).(fieldname),true_se_eccentricity(r
ep).(fieldname),true_CI_eccentricity(rep).(fieldname)] =
Network_metrics(temp_true_sig,true_freq_filter,pop_size,0,2.262);
    [true_louvain_ind, true_comm(rep).(fieldname), true_modularity(rep).(fieldname),
true_num_comm(rep).(fieldname), true_num_levels(rep).(fieldname)] =
Louvain_memory(true_ai(rep).(fieldname),rep,0,'period_1',i,simulation_no,no_louvain_repl
icates); %0 for true data
    filt_true(rep).(fieldname) = length(true_louvain_ind);
    [sample_ai(rep).(fieldname), ~,
sample_no_inds_in_sig_data(rep).(fieldname),sample_ave_sig_filt(rep).(fieldname),sample
_no_AIs(rep).(fieldname),sample_ave_AI_filt(rep).(fieldname),sample_sd_AI_filt(rep).(fiel
dname),sample_skew_AI_filt(rep).(fieldname),sample_kurtosis_AI_filt(rep).(fieldname),sa
mple_ave_deg(rep).(fieldname),sample_sd_deg(rep).(fieldname),sample_ave_weighted_deg
(rep).(fieldname),sample_sd_weighted_deg(rep).(fieldname),sample_density(rep).(fieldnam
e),sample_ave_CC(rep).(fieldname),sample_sd_CC(rep).(fieldname),sample_se_CC(rep).(fi
eldname),sample_CI_CC(rep).(fieldname),sample_ave_path_length(rep).(fieldname),sampl
e_sd_path_length(rep).(fieldname),sample_se_path_length(rep).(fieldname),sample_CI_pat
h_length(rep).(fieldname),sample_diameter(rep).(fieldname),sample_ave_eccentricity(rep).(
fieldname),sample_sd_eccentricity(rep).(fieldname),sample_se_eccentricity(rep).(fieldname
),sample_CI_eccentricity(rep).(fieldname)] =
Network_metrics(temp_sample_sig,sample_freq_filter,pop_size,1,2.262);

```

```

    [sample_louvain_ind, sample_comm(rep).(fieldname),
    sample_modularity(rep).(fieldname), sample_num_comm(rep).(fieldname),
    sample_num_levels(rep).(fieldname)] =
    Louvain_memory(sample_ai(rep).(fieldname),rep,1,'period_1',i,simulation_no,no_louvain_r
    eplicates); %0 for sample data
    filt_sample(rep).(fieldname) = length(sample_louvain_ind); %number of individuals
    that pass through the frequency filter for each replicate and each period
    temp_true_sig = [];
    temp_sample_sig = [];
end

```

```

true_sig_all_rep(rep).r = true_sig;
true_sig = zeros(pop_size*time,pop_size);
sample_sig_all_rep(rep).r = sample_sig;
sample_sig = zeros(pop_size*time,pop_size);
true_no_gp_all_t_all_rep(rep).r = true_no_gp_all_t;
true_no_gp_all_t = zeros(1,time);
sample_no_gp_all_t_all_rep(rep).r = sample_no_gp_all_t;
sample_no_gp_all_t = zeros(1,time);
index = 1; %counter to assign values to true sighting data
index2 = 1; %counter to assign values to sample sighting data
gp_id_all_t_all_rep(rep).r = gp_id_all_t;
gp_sizes_all_t_all_rep(rep).r = gp_sizes_all_t;
gp_sizes_all_t = zeros(time, pop_size);
gp_id_all_t = zeros(time, pop_size);
exp_gp_size = zeros(time, pop_size);
cuml_adj = zeros(pop_size, pop_size);

```

```
end % end of replicates loop
```

```

ave_incr_new_assoc = New_associates(true_ai, no_replicates,period_2,
pop_size,simulation_no);
xlswrite(filename, {'time','mean for all reps'},'Cumulative_ai','a1');
xlswrite(filename, {'var for all reps'},'Cumulative_ai','m1');
xlswrite(filename, {'std for all reps'},'Cumulative_ai','x1');
xlswrite(filename, [1:time],'Cumulative_ai','a2');
xlswrite(filename, cuml_ai_mean,'Cumulative_ai','b2');
xlswrite(filename, cuml_ai_var,'Cumulative_ai','m2');
xlswrite(filename, cuml_ai_std,'Cumulative_ai','x2');
xlswrite(filename, {'time','mean for all reps'},'Cumulative_adj','a1');
xlswrite(filename, {'var for all reps'},'Cumulative_adj','m1');
xlswrite(filename, {'std for all reps'},'Cumulative_adj','x1');
xlswrite(filename, [1:time],'Cumulative_adj','a2');
xlswrite(filename, cuml_adj_mean,'Cumulative_adj','b2');
xlswrite(filename, cuml_adj_var,'Cumulative_adj','m2');
xlswrite(filename, cuml_adj_std,'Cumulative_adj','x2');

```

```
ticktock1 = toc;
```

```
tic
```

```
xlswrite(strcat('Group_sizes_',num2str(simulation_no),'.xlsx'),
[1:time'],'Ave_group_sizes','a2');
xlswrite(strcat('Group_sizes_',num2str(simulation_no),'.xlsx'),{'time/replicates'},'Ave_group
_sizes','a1');
xlswrite(strcat('Group_sizes_',num2str(simulation_no),'.xlsx'),1:no_replicates,'Ave_group_s
izes','b1');
xlswrite(strcat('Group_sizes_',num2str(simulation_no),'.xlsx'),ave_gp_size,'Ave_group_size
s','b2');
```

```
xlswrite(strcat('Group_sizes_',num2str(simulation_no),'.xlsx'),
[1:time'],'Median_group_sizes','A2');
xlswrite(strcat('Group_sizes_',num2str(simulation_no),'.xlsx'),{'time/replicates'},'Median_gr
oup_sizes','a1');
xlswrite(strcat('Group_sizes_',num2str(simulation_no),'.xlsx'),1:no_replicates,'Median_grou
p_sizes','b1');
xlswrite(strcat('Group_sizes_',num2str(simulation_no),'.xlsx'),med_gp_size,'Median_group
_sizes','b2');
```

```
xlswrite(strcat('Group_sizes_',num2str(simulation_no),'.xlsx'),
[1:time'],'Ave_exp_group_sizes','A2');
xlswrite(strcat('Group_sizes_',num2str(simulation_no),'.xlsx'),{'time/replicates'},'Ave_exp_
group_sizes','a1');
xlswrite(strcat('Group_sizes_',num2str(simulation_no),'.xlsx'),1:no_replicates,'Ave_exp_gro
up_sizes','b1');
xlswrite(strcat('Group_sizes_',num2str(simulation_no),'.xlsx'),ave_exp_gp_size,'Ave_exp_g
roup_sizes','b2');
```

```
xlswrite(strcat('Group_sizes_',num2str(simulation_no),'.xlsx'),
[1:time'],'Var_group_sizes','A2');
xlswrite(strcat('Group_sizes_',num2str(simulation_no),'.xlsx'),{'time/replicates'},'Var_group
_sizes','a1');
xlswrite(strcat('Group_sizes_',num2str(simulation_no),'.xlsx'),1:no_replicates,'Var_group_si
zes','b1');
xlswrite(strcat('Group_sizes_',num2str(simulation_no),'.xlsx'),var_gp_size,'Var_group_size
s','b2');
```

```
xlswrite(strcat('Group_sizes_',num2str(simulation_no),'.xlsx'),[1:no_replicates'],'Var_ave_gr
oup_sizes','a2');
xlswrite(strcat('Group_sizes_',num2str(simulation_no),'.xlsx'),{'across all
t'},'Var_ave_group_sizes','b1');
xlswrite(strcat('Group_sizes_',num2str(simulation_no),'.xlsx'),var_ave_gp_size,'Var_ave_gr
oup_sizes','b2');
```

```
xlswrite(strcat('Group_sizes_',num2str(simulation_no),'.xlsx'),{'individuals/replicates'},'Ave
_exp_group_sizes_2','a1');
xlswrite(strcat('Group_sizes_',num2str(simulation_no),'.xlsx'),1:no_replicates,'Ave_exp_gro
up_sizes_2','b1');
xlswrite(strcat('Group_sizes_',num2str(simulation_no),'.xlsx'),[1:pop_size'],'Ave_exp_group
_sizes_2','a2');
```

```
xlswrite(strcat('Group_sizes_',num2str(simulation_no),'.xlsx'),ave_exp_gp_size_2,'Ave_exp_group_sizes_2','b2');
```

```
xlswrite(strcat('Group_sizes_',num2str(simulation_no),'.xlsx'),{'individuals/replicates'},'Var_exp_group_sizes','a1');  
xlswrite(strcat('Group_sizes_',num2str(simulation_no),'.xlsx'),1:no_replicates,'Var_exp_group_sizes','b1');  
xlswrite(strcat('Group_sizes_',num2str(simulation_no),'.xlsx'),[1:pop_size],'Var_exp_group_sizes','a2');  
xlswrite(strcat('Group_sizes_',num2str(simulation_no),'.xlsx'),var_exp_gp_size,'Var_exp_group_sizes','b2');
```

```
for i = 0:period_1  
    if i~=1  
        fieldname = strcat('period_1_',num2str(i));  
        for j = 1:no_replicates %this loop is there to get values of different replicates but same period as one list  
            true_ave_sig_filt_temp(j) = true_ave_sig_filt(j).(fieldname);  
            true_no_inds_in_sig_data_temp(j)=true_no_inds_in_sig_data(j).(fieldname);  
            true_no_AIs_temp(j)=true_no_AIs(j).(fieldname);  
            true_ave_AI_filt_temp(j)=true_ave_AI_filt(j).(fieldname);  
            true_sd_AI_filt_temp(j)=true_sd_AI_filt(j).(fieldname);  
            true_skew_AI_filt_temp(j)=true_skew_AI_filt(j).(fieldname);  
            true_kurtosis_AI_filt_temp(j)=true_kurtosis_AI_filt(j).(fieldname);  
            true_ave_deg_temp(j)=true_ave_deg(j).(fieldname);  
            true_sd_deg_temp(j)=true_sd_deg(j).(fieldname);  
            true_ave_weighted_deg_temp(j)=true_ave_weighted_deg(j).(fieldname);  
            true_sd_weighted_deg_temp(j)=true_sd_weighted_deg(j).(fieldname);  
            true_density_temp(j)=true_density(j).(fieldname);  
            true_ave_CC_temp(j)=true_ave_CC(j).(fieldname);  
            true_sd_CC_temp(j)=true_sd_CC(j).(fieldname);  
            true_se_CC_temp(j)=true_se_CC(j).(fieldname);  
            true_CI_CC_temp(j)=true_CI_CC(j).(fieldname);  
            true_ave_path_length_temp(j)=true_ave_path_length(j).(fieldname);  
            true_sd_path_length_temp(j)=true_sd_path_length(j).(fieldname);  
            true_se_path_length_temp(j)=true_se_path_length(j).(fieldname);  
            true_CI_path_length_temp(j)=true_CI_path_length(j).(fieldname);  
            true_diameter_temp(j)=true_diameter(j).(fieldname);  
            true_ave_eccentricity_temp(j)=true_ave_eccentricity(j).(fieldname);  
            true_sd_eccentricity_temp(j)=true_sd_eccentricity(j).(fieldname);  
            true_se_eccentricity_temp(j)=true_se_eccentricity(j).(fieldname);  
            true_CI_eccentricity_temp(j)=true_CI_eccentricity(j).(fieldname);  
  
            sample_ave_sig_filt_temp(j) = sample_ave_sig_filt(j).(fieldname);  
            sample_no_inds_in_sig_data_temp(j)=sample_no_inds_in_sig_data(j).(fieldname);  
            sample_no_AIs_temp(j)=sample_no_AIs(j).(fieldname);  
            sample_ave_AI_filt_temp(j)=sample_ave_AI_filt(j).(fieldname);  
            sample_sd_AI_filt_temp(j)=sample_sd_AI_filt(j).(fieldname);  
            sample_skew_AI_filt_temp(j)=sample_skew_AI_filt(j).(fieldname);  
            sample_kurtosis_AI_filt_temp(j)=sample_kurtosis_AI_filt(j).(fieldname);
```

```

sample_ave_deg_temp(j)=sample_ave_deg(j).(fieldname);
sample_sd_deg_temp(j)=sample_sd_deg(j).(fieldname);
sample_ave_weighted_deg_temp(j)=sample_ave_weighted_deg(j).(fieldname);
sample_sd_weighted_deg_temp(j)=sample_sd_weighted_deg(j).(fieldname);
sample_density_temp(j)=sample_density(j).(fieldname);
sample_ave_CC_temp(j)=sample_ave_CC(j).(fieldname);
sample_sd_CC_temp(j)=sample_sd_CC(j).(fieldname);
sample_se_CC_temp(j)=sample_se_CC(j).(fieldname);
sample_CI_CC_temp(j)=sample_CI_CC(j).(fieldname);
sample_ave_path_length_temp(j)=sample_ave_path_length(j).(fieldname);
sample_sd_path_length_temp(j)=sample_sd_path_length(j).(fieldname);
sample_se_path_length_temp(j)=sample_se_path_length(j).(fieldname);
sample_CI_path_length_temp(j)=sample_CI_path_length(j).(fieldname);
sample_diameter_temp(j)=sample_diameter(j).(fieldname);
sample_ave_eccentricity_temp(j)=sample_ave_eccentricity(j).(fieldname);
sample_sd_eccentricity_temp(j)=sample_sd_eccentricity(j).(fieldname);
sample_se_eccentricity_temp(j)=sample_se_eccentricity(j).(fieldname);
sample_CI_eccentricity_temp(j)=sample_CI_eccentricity(j).(fieldname);

```

```
end
```

```
%
```

```

Write_network_stats(filename,0,'period_1',i,no_replicates,pop_size,time,ave_gp_size,memory_length,true_ave_sig_filt_temp,true_no_inds_in_sig_data_temp,true_no_AIs_temp,true_ave_AI_filt_temp,true_sd_AI_filt_temp,true_skew_AI_filt_temp,true_kurtosis_AI_filt_temp,true_ave_deg_temp,true_sd_deg_temp,true_ave_weighted_deg_temp,true_sd_weighted_deg_temp,true_density_temp,true_ave_CC_temp,true_sd_CC_temp,true_se_CC_temp,true_CI_CC_temp,true_ave_path_length_temp,true_sd_path_length_temp,true_se_path_length_temp,true_CI_path_length_temp,true_diameter_temp,true_ave_eccentricity_temp,true_sd_eccentricity_temp,true_se_eccentricity_temp,true_CI_eccentricity_temp);

```

```

Write_network_stats(filename,1,'period_1',i,no_replicates,pop_size,time,ave_gp_size,memory_length,sample_ave_sig_filt_temp,sample_no_inds_in_sig_data_temp,sample_no_AIs_temp,sample_ave_AI_filt_temp,sample_sd_AI_filt_temp,sample_skew_AI_filt_temp,sample_kurtosis_AI_filt_temp,sample_ave_deg_temp,sample_sd_deg_temp,sample_ave_weighted_deg_temp,sample_sd_weighted_deg_temp,sample_density_temp,sample_ave_CC_temp,sample_sd_CC_temp,sample_se_CC_temp,sample_CI_CC_temp,sample_ave_path_length_temp,sample_sd_path_length_temp,sample_se_path_length_temp,sample_CI_path_length_temp,sample_diameter_temp,sample_ave_eccentricity_temp,sample_sd_eccentricity_temp,sample_se_eccentricity_temp,sample_CI_eccentricity_temp);

```

```
end
```

```
end
```

```
for i = 0:period_1
```

```
if i~=1
```

```

Write_louvain_results(filename,0,'period_1',i,no_replicates,pop_size,no_louvain_replicates,true_num_levels,true_modularity,true_num_comm, strcat('period_1_',num2str(i)));

```

```

Write_louvain_results(filename,1,'period_1',i,no_replicates,pop_size,no_louvain_replicates,
sample_num_levels,sample_modularity,sample_num_comm,
strcat('period_1_',num2str(i)));

```

```

    end
end

```

```

group_size_summary(1,1:no_replicates) = mean(ave_gp_size,2);
group_size_summary(2,1:no_replicates) = mean(med_gp_size,2);
group_size_summary(3,1:no_replicates) = mean(var_gp_size,2);
group_size_summary(4,1:no_replicates) = var_ave_gp_size;
group_size_summary(5,1:no_replicates) = mean(ave_exp_gp_size,2);
group_size_summary(6,1:no_replicates) = mean(ave_exp_gp_size_2,2);
group_size_summary(7,1:no_replicates) = mean(var_exp_gp_size,2);
xlswrite(filename, {'replicates'}, 'Group_sizes', 'a1');
xlswrite(filename, 1:no_replicates, 'Group_sizes', 'b1');
xlswrite(filename, {'Ave. of Ave. group size (over time)'; 'Ave. of Median group size(over
time)'; 'Ave. variance in group size every time step'; 'Variance in ave group size with time';
'Ave. of ave. (over inds) experienced group size (over time)'; 'Ave. of ave. (over time)
experienced group size 2 (ave over individuals)'; 'Ave. variance (over time) in experienced
group size with time (ave over individuals)'}, 'Group_sizes', 'a2')
xlswrite(filename, group_size_summary, 'Group_sizes', 'b2')

```

```

ticktock2 = toc;

```

Group_sizes.m

```

function gp_sizes =
Group_sizes(gp_var_type,gp_size_type,pop_size,t,original_gp_sizes_all_t)
if gp_size_type ==2
    no_success = 1;
    prob_success = 0.5;
elseif gp_size_type == 4
    no_success = 4;
    prob_success = 0.5;
end
if t==1
    flag = 1;
else
    flag = 0;
end

if gp_var_type==0
    gp_sizes = zeros(1,pop_size);
    if gp_size_type ==2
        %create groups for every time step, group sizes add up to population size.
        %This is a simplest case with group sizes that are equal and fixed across all time steps
        and all groups

```

```

    no_gp = 100;
elseif gp_size_type ==4
    no_gp = 50;
end
    temp = pop_size/no_gp;
    gp_sizes(1:no_gp) = repelem(temp,no_gp);
elseif gp_var_type==1
    gp_sizes = zeros(1,pop_size);
%assign group sizes based on negative binomial distribution. Ensure group sizes add up to
population size.
    c =1;
    while(sum(gp_sizes)~=pop_size)
        temp = nbinrnd(no_success,prob_success);
        if temp>=1 %to get non zero group sizes
            gp_sizes(c) = temp;
            c = c+1;
        end

        if sum(gp_sizes)>pop_size
            gp_sizes = zeros(1,pop_size);
            c = 1;
        elseif sum(gp_sizes)<pop_size
            continue;
        else
            if gp_size_type==2
                if mean(gp_sizes(gp_sizes>0))>=1.5 & mean(gp_sizes(gp_sizes>0))<=2.5
                    break;
                else
                    gp_sizes = zeros(1,pop_size);
                    c = 1;
                end
            elseif gp_size_type==4
                if mean(gp_sizes(gp_sizes>0))>=3.5 & mean(gp_sizes(gp_sizes>0))<=4.5
                    break;
                else
                    gp_sizes = zeros(1,pop_size);
                    c = 1;
                end
            end
        end
    end
end
end
end
end
end

```

Adjacency_matrix.m

```
function adj_every_t = Adjacency_matrix(gp_id, pop_size)
adj_every_t = zeros(pop_size, pop_size);
for i = 1:pop_size
    for j = 1:pop_size
        if i~=j
            if gp_id(i)==gp_id(j)
                adj_every_t(i,j) = 1;
            end
        end
    end
end
end
end
```

Separate_sighting.m

```
function sighting_str = Separate_sighting(no_gp_all_t, s,time, period_1, period_2)
incr = time/period_1;    %should always be a whole number
k = 1;
l = 0;
dummy = 1;
while (l<time)
    l = l+incr;
    sum_gps(dummy) = sum(no_gp_all_t(k:l));
    k = l+1 ;
    dummy = dummy+1;
end
k = 0;
l = 0;
dummy = 1;
while(l<time)
    l = l+incr;
    fieldname = strcat('period_1_',num2str(dummy));
    sighting_str(fieldname) = s(k+1:k+sum_gps(dummy),:);
    k = k+sum_gps(dummy);
    dummy = dummy+1;
end

incr = time/period_2;    %should always be a whole number
k = 1;
l = 0;
dummy = 1;
while (l<time)
    l = l+incr;
    sum_gps(dummy) = sum(no_gp_all_t(k:l));
    k = l+1;
    dummy = dummy+1;
```

```

end
k = 0;
l = 0;
dummy = 1;
while(l<time)
    l = l+incr;
    fieldname = strcat('period_2_',num2str(dummy));
    sighting_str.(fieldname) = s(k+1:k+sum_gps(dummy),:);
    k = k+sum_gps(dummy);
    dummy = dummy+1;
end

end

```

Network_metrics.m

```

function [AI_filt_sym, Sig,
no_inds_in_sig_data,ave_sig_filt,no_AIs,ave_AI_filt,sd_AI_filt,skew_AI_filt,kurtosis_AI_f
ilt,ave_deg, sd_deg,ave_weighted_deg,
sd_weighted_deg,density,ave_CC,sd_CC,se_CC,CI_CC,ave_path_length,sd_path_length,se
_path_length,CI_path_length,diameter,ave_eccentricity,sd_eccentricity,se_eccentricity,CI_e
ccentricity] = Network_metrics(sig_data,freq_filter, no_inds,sample,t_val)
    inp = sort(sig_data,2,'ascend');
    uniqID=no_inds;
    Sig=zeros(uniqID,1);
    Assoc=zeros(uniqID);
    for i=1:size(inp,1) %This code block counts the number of times each individual is
seen in the overall sighting data
        for k=1:size(inp,2)
            if inp(i,k)>0
                if inp(i,k)<(uniqID+1)
                    Sig(inp(i,k))= Sig(inp(i,k))+1;
                end
            end
        end
    end
    filtered_uniqID=0;
    for i=1:uniqID %This code block counts the number of individuals which pass the
frequency filter
        if (Sig(i)>freq_filter)
            filtered_uniqID=filtered_uniqID+1;
        end
    end
    filter_select=1:uniqID; %in the loop below assigned zero if the sightings of that
individual is not retained, if retained the variable takes the value of the number of sightings
of that individual

```

```

if (i<uniqID)
    for j=(i+1):uniqID
        if ((Sig(i)+Sig(j))>0 && filter_select(i)>0 && filter_select(j)>0)

AI_filtered(mapping_inv(i),mapping_inv(j))=Assoc_filtered(mapping_inv(i),mapping_inv(j)
))/(Sig(i)+Sig(j)-Assoc_filtered(mapping_inv(i),mapping_inv(j)));
        end
    end
end
end

Assoc_filt_size=size(Assoc_filtered,1);
AI_filt_sym=AI_filtered + AI_filtered';

if sample==1
    AI_sample = zeros(no_inds,no_inds);
    for i = 1:no_inds
        if i<no_inds
            for j = (i+1):no_inds
                if mapping_inv(i) == 0
                    AI_sample(i,j) = 0;
                else
                    if mapping_inv(j) == 0
                        AI_sample(i,j) = 0;
                    else
                        AI_sample(i,j) = AI_filtered(mapping_inv(i),mapping_inv(j));
                    end
                end
            end
        end
    end
end
end
AI_filt_sym = [];
AI_filt_sym = AI_sample+ AI_sample' ;
end

ones_matrix=ones(size(AI_filt_sym));
ones_matrix=tril(ones_matrix,-1);
AI_filt_sym_list=AI_filt_sym(find(ones_matrix==1)); %this is the list of all association
indices, basically the upper triangle data in the form of a list
no_inds_in_sig_data= size(unique(sig_data),1)-1 ; % -1 is to take out the zeros.

ave_sig_filt= sum(Sig_filtered)/no_inds_in_sig_data;
no_AIs= nchoosek(no_inds_in_sig_data,2);
diff_AI_nos=size(AI_filt_sym_list,1) - no_AIs;

ave_AI_filt=mean(AI_filt_sym_list);
sd_AI_filt=std(AI_filt_sym_list);
skew_AI_filt=skewness(AI_filt_sym_list);
kurtosis_AI_filt=kurtosis(AI_filt_sym_list);

```

```

adjacency_matrix=zeros(size(AI_filt_sym));
x=find(AI_filt_sym>0);
adjacency_matrix(x)=1;

% To find the degree of vertices %%%%%%%%%%for upper triangle
degree=zeros(no_inds,1);
for i=1:no_inds
    for j=1:i
        if(AI_filt_sym(j,i)>0)
            degree(i)=degree(i)+1;
        end
    end
    for k=i:no_inds %%%%%%%%%% for lower triangle
        if(AI_filt_sym(i,k)>0)
            degree(i)=degree(i)+1;
        end
    end
end
% diff_ind_nos=no_inds - no_inds_in_sig_data; %degree not calculated if no of inds in
% sig data is less than actual number of inds
% if (diff_ind_nos > 0)
%     temp_degree=sort(degree,1);
%     clear degree;
%     degree=temp_degree(diff_ind_nos+1:end,:);
% end

ave_deg=mean(degree);
sd_deg=std(degree);

% To find the weighted degree of vertices %%%%%%%%%%for upper triangle
weighted_degree=zeros(no_inds,1);
for i=1:no_inds
    for j=1:i
        if(AI_filt_sym(j,i)>0)
            weighted_degree(i)= AI_filt_sym(j,i)+ weighted_degree(i);
        end
    end
    for k=i:no_inds %%%%%%%%%% for lower triangle
        if(AI_filt_sym(i,k)>0)
            weighted_degree(i)= AI_filt_sym(i,k)+ weighted_degree(i);
        end
    end
end
end

% diff_ind_nos=no_inds - no_inds_in_sig_data;
% if (diff_ind_nos > 0)
%     temp_weighted_degree=sort(weighted_degree,1);
%     clear weighted_degree;
%     weighted_degree=temp_weighted_degree(diff_ind_nos+1:end,:);

```

```

% end

ave_weighted_deg=mean(weighted_degree);
sd_weighted_deg=std(weighted_degree);

% GETTING NO. OF EDGES AND DENSITY
count_assoc=0; % This gives the total number of associations (edges) across
all individual pairs (not the total value of associations).
for i=1:no_inds
    for j=1:i
        if(AI_filt_sym(i,j)>0)
            count_assoc=count_assoc+1;
        end
    end
end
%possible_edges=filtered_uniqID * (filtered_uniqID-1) / 2;
possible_edges=no_inds * (no_inds-1) / 2; % This is because, while randomly picking up
inds, some inds may not turn up in the sighting data at all.
density = count_assoc/possible_edges;

% CLUSTERING COEFFICIENT
graph=adjacency_matrix;

no_triangles = diag(graph*triu(graph)*graph); % Number of triangles for each node as
opposed to total number of triangles - see above.

% The local clustering coefficient of each node.
CC_node = zeros(size(degree));
CC_node(degree > 1) = 2 * no_triangles(degree > 1) ./ (degree(degree >
1).*(degree(degree > 1) - 1));

%Average clustering coefficient of the graph
ave_CC = mean(CC_node(degree > 1));
sd_CC = std(CC_node(degree > 1));
se_CC = sd_CC/sqrt(size(CC_node(degree>1),1));
CI_CC = t_val*se_CC;

% PATH LENGTHS

path_lengths=inf*ones(length(adjacency_matrix)); % Because if there is no connection,
the path length will be infinity.
node_nos = 1:length(adjacency_matrix); % These are node nos for which the path lengths
are not yet found.
for i=1:length(adjacency_matrix)
    path_lengths(i,i)=0; % Distance of node with itself.
    node_nos = 1:length(adjacency_matrix); % Nodes with path lengths not found.
    while not isempty(node_nos)
        [min_path_length,index] = min(path_lengths(i,node_nos)); % [Y,I] = MIN(X) gives
Y with the min value and I containing the indices of the minimum values. If there is more
than one minimal element, the index of the first one is returned.

```

```

    % The above should give the index of the focal node from which path lengths are
    being calculated.
    for j=1:length(node_nos)
        if adjacency_matrix(node_nos(index),node_nos(j))>0 &
path_lengths(i,node_nos(j))>path_lengths(i,node_nos(index))+adjacency_matrix(node_nos(
index),node_nos(j));

path_lengths(i,node_nos(j))=path_lengths(i,node_nos(index))+adjacency_matrix(node_nos(
index),node_nos(j));
        end
    end
    node_nos = setdiff(node_nos,node_nos(index)); % SETDIFF(A,B) when A and B
are vectors returns the values in A that are not in B. The result will be sorted.
    end
end

% when_no_paths=find(path_lengths_1==Inf);
no_of_shortest_paths=length(find(path_lengths<Inf))-length(adjacency_matrix);

ones_matrix=ones(size(path_lengths));
ones_matrix=tril(ones_matrix,-1);
path_lengths_list=path_lengths(find(ones_matrix==1));
ave_path_length=mean(path_lengths_list(path_lengths_list<Inf)); % Taking mean of
those path lengths that are not Infinity in value.
sd_path_length=std(path_lengths_list(path_lengths_list<Inf));
se_path_length=sd_path_length/sqrt(size(path_lengths_list(path_lengths_list<Inf),1));
CI_path_length=t_val*se_path_length;

diameter=max(path_lengths_list(path_lengths_list<Inf));

% ECCENTRICITY (this is the longest of the shortest paths from each node)
temp2=path_lengths; % This is to overcome the problem of having nodes with no
connections and, therefore, infinity path lengths.
temp2(temp2==Inf)=NaN; % By default, NaN will be omitted, so the infinity values will
be omitted when replaced by NaN.
temp2(temp2==0)=NaN;
eccentricity=max(temp2,[],2); % [Y,I] = MAX(X,[],DIM) operates along the dimension
DIM.
ave_eccentricity=mean(eccentricity(eccentricity<Inf));
sd_eccentricity=std(eccentricity(eccentricity<Inf));
se_eccentricity=sd_eccentricity/sqrt(size(eccentricity(eccentricity<Inf),1));
CI_eccentricity=t_val*se_eccentricity;
end

```

Louvain_memory.m

```
function [ind,s, mod_max,numComm, levels] = louvain(ai,replicate,sample, period_type,  
per,simulation_no,rep)
```

```
%% This code block is taken from Kabini_assoc_from_excel_file_mod_2020 and modified  
slightly. It creates a text file with edges and weights using the AI matrix  
count_assoc=0; % This gives the total number of associations across all  
individual pairs (not the total value of associations).
```

```
for i=1:size(ai,1)  
    for j=1:size(ai,1)  
        if(ai(i,j)>0)  
            count_assoc=count_assoc+1;  
        end  
    end  
end  
edges=ones(count_assoc,3);  
temp_index=1;  
for i=1:size(ai,1)  
    for j=1:size(ai,1)  
        if(ai(i,j)>0)  
            edges(temp_index,1)=i;  
            edges(temp_index,2)=j;  
            edges(temp_index,3)= ai(i,j);  
            temp_index=temp_index+1;  
        end  
    end  
end  
newfilename=strcat('Edge_list','_',num2str(simulation_no),'_', num2str(replicate),'_',  
num2str(sample),'_',period_type,'_', num2str(per),'_','csv');  
dlmwrite(newfilename, edges, 'delimiter', ' ', 'newline', 'pc');
```

```
%% This code block is taken from multiCpp.modA and has been modified. It calls the C++  
executables of the louvain algorithm. Communities obtained in each pass is sent to  
modularity func to calculate max modularity attained in that pass.
```

```
cppConvert= 'D:\Anvitha\Louvain_v0.3_core1\gen-louvain\convert'; %path of the C++  
executables in the computer. Ensure that the required .dll files from cygwin bin are present  
in the folder with these .exe files  
cppHierarchy='D:\Anvitha\Louvain_v0.3_core1\gen-louvain\hierarchy';  
cppCommunity='D:\Anvitha\Louvain_v0.3_core1\gen-louvain\louvain';  
binFile='edges.bin'; %file with output from convert.exe  
weightsFile='edges.weights'; %file with output from convert.exe  
treeFile='edges.tree'; %file with output from louvain.exe  
edgesFile= newfilename; %input file with list of edges and weight  
s = struct;  
mod_max = struct;  
numComm = struct;  
levels = zeros(rep,1);
```

```

for i = 1:rep
    commandLine = sprintf('%s -i %s -o %s -w %s ',cppConvert,
edgesFile,binFile,weightsFile); %to call convert.exe from command line, input is edges file,
output is edges.bin and edges.weights file
    [a, b] = system(commandLine);
    commandLine=sprintf('%s %s -l -l -q %s -w %s > %s',cppCommunity,
binFile,'0',weightsFile,treeFile); %to call louvain.exe, input is edges.bin and edges.weights
while the output is edges.tree which has list of communities for all the passes, '0' as input
uses Newman-Girvan modularity as quality function
    [dummy1 dummy2]= system(commandLine);
    commandLine=sprintf('%s %s ',cppHierarchy, treeFile); %to call hierarchy.exe to obtain
a list of number of communities in each pass, input is edges.tree file
    [dummy1 dummy2]= system(commandLine);
    dummy3=strread(dummy2,'%s','delimiter',':');
    numLevels=str2num(dummy3{2}) ;
    levels(i,1) = numLevels;
    for j=0:numLevels-1 %this code block separates communities of different passes present
in edges.tree into different text files
        nodesFile='nodes';
        nodesFile=[nodesFile '_' num2str(j),'.txt' ];
        commandLine=sprintf('%s %s -l %d > %s',cppHierarchy, treeFile,j,nodesFile);
        fprintf('%s\n', commandLine);
        [dummy1 dummy2]=system(commandLine);
        t = readtable(nodesFile); %converting text file with communities into a table
        fieldname = strcat('pass',num2str(j)); %to name field for structure s, each pass is one
field
        ind = table2array(t(2:end,1));
        s(i).(fieldname) = table2array(t(2:end,2)); %adding communities of a given pass of ith
iteration into a structure
        communities = table2array(t(2:end,2));
        numComm(i).(fieldname) = max(communities);
        mod_max(i).(fieldname) = Modularity(ai,ind,communities);
        delete (nodesFile);
    end
    delete (binFile);
    delete (weightsFile);
    delete (treeFile);
end

if replicate ~= 2 && replicate ~= 3
    delete (edgesFile)
else
    newfilename=strcat('Edge_list','_',num2str(simulation_no),'_', num2str(replicate),'_',
num2str(sample),'_',period_type,'_', num2str(per),'.net');
    dlmwrite(newfilename, edges, 'delimiter', ' ', 'newline', 'pc');
    delete (edgesFile);
end
end

```

Modularity.m

```
function Q = modularity(ai,ind,comm)
format long;
m = sum(sum(ai));
Q = 0;
sum_i = 0;
ai = double(ai);
for i = 1:length(ind)
    sum_j = 0;
    for j = 1:length(ind)

        Aij = ai(i,j);
        ki = sum(sum(ai(i,:)));
        kj = sum(sum(ai(j,:)));
        if isequal(comm(i,1),comm(j,1))
            delta = 1;
        else
            delta = 0;
        end
        sum_j = sum_j + (Aij - (ki*kj/m))*delta;

    end
    sum_i = sum_i + sum_j;
end
Q = sum_i/m;
end
```

New_associates.m

```
function ave_deg_incr = New_associates(true_ai, no_replicates, period,
pop_size, simulation_no)
uni = zeros(pop_size, period);
ave_deg_incr = zeros(no_replicates, period);
deg_incr = struct;
for i = 1:no_replicates
    a1 = true_ai(i).('period_2_1')>0;
    for k = 1:pop_size
        b1 = find(a1(k, :)==1);
        uni(k, 1) = length(b1);
        for j = 2:period
            fieldname = strcat('period_2_', num2str(j));
            a2 = true_ai(i).(fieldname)>0;
            b2 = find(a2(k, :)==1);
            dummy = setdiff(b2(1, :), b1(1, :));
            dummy2 = length(dummy);
            uni(k, j) = uni(k, j-1) + dummy2;
            b1 = unique([b1 b2]);
        end
    end
    deg_incr(i).rep = uni;
%     dummy3 = true_ai(i).('period_1_0')>0;
%     for k = 1:pop_size
%         ind_degree(i, k) = sum(dummy3(k, :)>0);
%     end
    ave_deg_incr(i, :) = mean(deg_incr(i).rep);

    xlswrite(strcat('Group_sizes_', num2str(simulation_no), '.xlsx'), ave_deg_incr(i, :), 'New_assoc
iates', strcat(CHAR(i+1), '2'));
end
xlswrite(strcat('Group_sizes_', num2str(simulation_no), '.xlsx'),
[1:period], 'New_associates', 'a2');
xlswrite(strcat('Group_sizes_', num2str(simulation_no), '.xlsx'), {'period/replicates'}, 'New_ass
ociates', 'a1');
xlswrite(strcat('Group_sizes_', num2str(simulation_no), '.xlsx'), 1:no_replicates, 'New_associat
es', 'b1');
end
```

Write_network_stats.m

```
function Write_network_stats(filename,sample,period_type,
per,no_replicates,pop_size,time,ave_gp_size,memory,ave_sig_filt,no_inds_in_sig_data,no_AIs,ave_AI_filt,sd_AI_filt,skew_AI_filt,kurtosis_AI_filt,ave_deg,sd_deg,
ave_weighted_deg,sd_weighted_deg,density,ave_CC,sd_CC,se_CC,CI_CC,
ave_path_length,sd_path_length,se_path_length,CI_path_length,diameter,
ave_eccentricity,sd_eccentricity,se_eccentricity,CI_eccentricity)
sheetname = strcat('Network_stats_', num2str(sample),'_',period_type,'_',num2str(per));
heading = {'No. of replicates' 'No. of inds' 'No. of time steps' 'Ave. gp. size' 'Memory type'
" " " 'Ave. no. of sig. filt' 'No. ind in sig data' 'No. of AIs' 'Ave. AI filt' 'SD AI filt' 'Skew in
AI filt' 'Kurtosis of AI filt' 'Ave. degree' 'SD degree' 'Ave. weighted degree' 'SD weighted
degree' 'Density' 'Ave. Clust Coeff' 'SD Clust Coeff' 'SE Clust Coeff' '95% CI Clust Coeff'
'Ave. path length' 'SD path length' 'SE path length' '95% CI path length' 'Diameter' 'Ave.
eccentricity' 'SD eccentricity' 'SE eccentricity' '95% CI eccentricity'};
xlswrite (filename, heading, sheetname,'A1');
xlswrite (filename, no_replicates, sheetname,'A2');
xlswrite (filename, pop_size, sheetname,'C2');
xlswrite (filename, time, sheetname,'D2');
xlswrite (filename, mean(ave_gp_size,2) , sheetname,'E2');
xlswrite (filename, memory , sheetname,'F2');
xlswrite (filename, ave_sig_filt, sheetname,'J2');
xlswrite (filename, no_inds_in_sig_data, sheetname,'K2');
xlswrite (filename, no_AIs, sheetname,'L2');
xlswrite (filename, ave_AI_filt, sheetname, 'M2');
xlswrite (filename, sd_AI_filt, sheetname, 'N2');
xlswrite (filename, skew_AI_filt, sheetname, 'O2');
xlswrite (filename, kurtosis_AI_filt, sheetname, 'P2');
xlswrite (filename, ave_deg, sheetname, 'Q2');
xlswrite (filename, sd_deg, sheetname, 'R2');
xlswrite (filename, ave_weighted_deg, sheetname, 'S2');
xlswrite (filename, sd_weighted_deg, sheetname, 'T2');
xlswrite (filename, density, sheetname, 'U2');
xlswrite (filename, ave_CC, sheetname, 'V2');
xlswrite (filename, sd_CC, sheetname, 'W2');
xlswrite (filename, se_CC, sheetname, 'X2');
xlswrite (filename, CI_CC, sheetname, 'Y2');
xlswrite (filename, ave_path_length, sheetname, 'Z2');
xlswrite (filename, sd_path_length, sheetname, 'AA2');
xlswrite (filename, se_path_length, sheetname, 'AB2');
xlswrite (filename, CI_path_length, sheetname, 'AC2');
xlswrite (filename, diameter, sheetname, 'AD2');
xlswrite (filename, ave_eccentricity, sheetname, 'AE2');
xlswrite (filename, sd_eccentricity, sheetname, 'AF2');
xlswrite (filename, se_eccentricity, sheetname, 'AG2');
xlswrite (filename, CI_eccentricity, sheetname, 'AH2');
end
```

Write_louvain_results.m

function

```
Write_louvain_results(filename,sample,period_type,per,no_replicates,pop_size,no_louvain_
replicates,num_levels,modularity,num_comm,fieldname)
sheetname = strcat('L_output_', num2str(sample),'_',period_type,'_',num2str(per));
heading = {'Data set replicate' 'Louvain replicate' 'No. of passes/levels' 'Modularity pass 0'
'Modularity pass 1' 'Modularity pass 2' 'Modularity pass 3' " " 'No. of communities pass 0'
'No. of communities pass 1' 'No. of communities pass 2' 'No. of communities pass 3'};
xlswrite(filename,heading,sheetname,'A1');
for i = 1:no_replicates
    k = i-1;
    xlswrite(filename,i,sheetname, strcat('a',num2str(no_louvain_replicates*k+2)));
    l = 1:no_louvain_replicates;
    xlswrite(filename,l,sheetname, strcat('b',num2str(no_louvain_replicates*k+2)));
    xlswrite(filename, num_levels(i).(fieldname), sheetname,
    strcat('c',num2str(no_louvain_replicates*k+2)));
    ave_comm_levels(i) = mean(num_levels(i).(fieldname));
    sd_comm_levels(i) = std(num_levels(i).(fieldname));
    n_comm_levels(i) = length(num_levels(i).(fieldname));
    table2 = struct2table(modularity(i).(fieldname));
    table3 = struct2table(num_comm(i).(fieldname));
    col = size(table2,2);
    for j = 1:col
        temp = table2array(table2(:,j)); %output is a cell array if there are a few empty rows ([])
in a given column of the table. Empty rows arise when a pass is not reached.
        if iscell(temp)
            ave_modularity(i,j) = mean(cell2mat(temp));
            sd_modularity(i,j) = std(cell2mat(temp));
            n_modularity(i,j) = length(cell2mat(temp));
        else
            ave_modularity(i,j) = mean(temp);
            sd_modularity(i,j) = std(temp);
            n_modularity(i,j) = length(temp);
        end
        xlswrite(filename, temp, sheetname,
    strcat(char('c'+j),num2str(no_louvain_replicates*k+2)))
    end
    col = size(table3,2);
    for j = 1:col
        temp2 = table2array(table3(:,j)); %output is a cell array if there are a few empty rows
([]) in a given column of the table. Empty rows arise when a pass is not reached.
        if iscell(temp2)
            ave_no_comm(i,j) = mean(cell2mat(temp2));
            sd_no_comm(i,j) = std(cell2mat(temp2));
            n_no_comm(i,j) = length(cell2mat(temp2));
        else
            ave_no_comm(i,j) = mean(temp2);
            sd_no_comm(i,j) = std(temp2);
            n_no_comm(i,j) = length(temp2);
```

```

    end
    xlswrite(filename, temp2, sheetname,
    strcat(char('c'+6+j),num2str(no_louvain_replicates*k+2))); %if there are more than 5
    passes, +6 must be changed to a higher number
    end
end

%%Remeber to edit excel column numbers in xlswrite statement if the number of passes is
more than 5
sheetname2 = strcat('Ave_L_output_', num2str(sample),'_',period_type,'_',num2str(per));
heading = {'Data set replicate' 'Ave. no. of passes/levels' 'SD of no. of passes/levels' 'n' '
'Ave. modularity pass 0' 'Ave. modularity pass 1' 'Ave. modularity pass 2' 'Ave. modularity
pass3' ' ' 'SD modularity pass 0' 'SD modularity pass 1' 'SD modularity pass 2' 'SD
modularity pass 3' ' ' 'n pass 0' 'n pass 1' 'n pass 2' 'n pass 3' ' ' 'Ave. no. of communities
pass 0' 'Ave. no. of communities pass 1' 'Ave. no. of communities pass 2' 'Ave. no. of
communities pass 3' ' ' 'SD no. of communities pass 0' 'SD no. of communities pass 1'
'SD no. of communities pass 2' 'SD no. of communities pass 3' ' ' 'n pass 0' 'n pass 1' 'n pass
2' 'n pass 3'};
xlswrite(filename,heading,sheetname2,'A1');
temp = 1:no_replicates;
xlswrite(filename,temp', sheetname2,'A2');
xlswrite(filename,ave_comm_levels',sheetname2,'B2');
xlswrite(filename,sd_comm_levels',sheetname2,'C2');
xlswrite(filename,n_comm_levels',sheetname2,'D2');
xlswrite(filename,ave_modularity,sheetname2,'F2');
xlswrite(filename,sd_modularity,sheetname2,'L2');
xlswrite(filename,n_modularity,sheetname2,'Q2');
xlswrite(filename,ave_no_comm,sheetname2,'W2');
xlswrite(filename,sd_no_comm,sheetname2,'AD2');
xlswrite(filename,n_no_comm,sheetname2,'AI2');

end

```