Modeling the Harvard Housing System

Ryan Song

December 21, 2015

Abstract

The Harvard housing system utilizes blocking groups in order to ensure that friends stay together when assigned to a house. However, creating blocking groups could ultimately lead to separation of friends if friend groups are too big. Using network models, hierarchical clustering, and Monte Carlo methods, we created a model of the housing system to determine the optimal blocking group size to maximize proximity of friends. The results of the simulations were surprising, but this was probably due to the assumptions and limitations of the model.

1 Introduction

Harvard College utilizes a unique way of assigning its students to their living spaces. After freshman year, each undergrad is placed into one of twelve houses that make up four neighborhoods. Before housing assignments are made, each freshman creates a blocking group consisting of 1 to 8 students. These blocking groups ensure that every member of the group ends up in the same house together. Additionally, two blocking groups can link with each other, which guarantees that the houses are in the same neighborhood, but also guarantees that they will not be in the same house together. After blocking and linking groups are created, each blocking group is then randomly sorted into one of the 12 houses, which is where the students live for the next three years.

Although there are many advantages that this system offers, one of the biggest problems of the housing system is the amount of drama and social tension that comes with creating blocking groups. Oftentimes, when friend circles exceed the blocking size limit of 8, they end up being forced to split into multiple groups or kick some people out of the group; if the groups do not end up linking together, this could result in friend groups that are geographically separated (e.g. one group ends up assigned to a river house while the other group gets "quadded"), which could ultimately lead to loss of contact with previously close friends.

As a result, it would be an interesting question to evaluate how effective the current housing system is, and whether there is a better way of organizing the housing system. Specifically, the goals that this paper seeks to answer are:

- Observe the size of friend circles in Harvard and determine whether a reevaluation of the housing policy is necessary.
- Find the optimal size of blocking groups that will minimize the geographical distance between friends.
- Identify and explore the assumptions and limitations of the model and understand how these could skew our results.

2 Model

2.1 Social Network Model

The first step is to model the friend circles that are present on campus. To accomplish, we used the Watts-Strogatz Model.[1] The Watts-Strogatz Model is a random graph that has small world properties, such as high clustering and short average geodesic paths.

The creation of the Watts-Strogatz Model requires three parameters: N (the number of nodes in the network), $2K$ (the mean degree of each node), and β (a probability parameter where $0 \leq \beta \leq 1$). The first step in creating the model is to start off with a regular lattice with N nodes, where each node is connected to the K neighbors on its right and left. The next step is to take every single connection in the network and rewire the node (except to itself) with a probability of β.

The advantages of using this model compared to other models is that it allows for small world properties that are characteristic of social networks. Previous models used, such as a regular lattice and a random graph, are not able to capture all the small world properties that the Watts-Strogatz Model does. By essentially taking aspects from both of these previous model, the Watts-Strogatz Model is able to capture both the small average geodesic path of random graphs and the clustering of a regular lattice network (see Figure 1 for graphical comparisons).

In order to model the social networks of the Harvard undergraduate population, the parameter values used were $N = 85$, $K = 4$, and $\beta = 0.01$. We chose an N value of 85 because higher values of N led to computational difficulties (explained further later). The reasoning behind choosing a K value of 4 is because we can assume that friends will choose

[1] `http://www.nature.com.ezp-prod1.hul.harvard.edu/nature/journal/v393/n6684/pdf/393440a0.pdf`

to block with only their closest friends, which usually includes around 5-10 friends.[2] The reasoning behind choosing a β value of 0.01 is because this value is supposed to maximize clustering while minimizing geodesic distance, which is desirable when modeling social networks.[3]

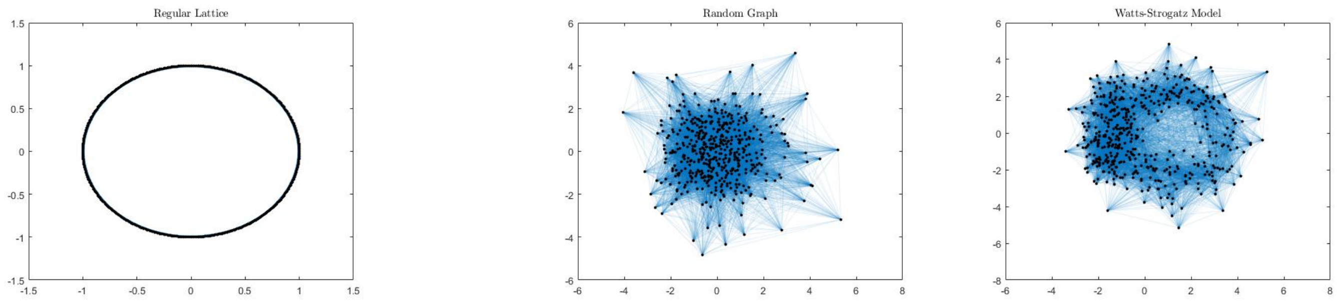

Figure 1: Three different models of social networks (where $N = 500$ and $K = 25$). The left graph is a Regular Lattice. The middle graph is a Random Graph. The Right graph is a Watts-Strogatz Model with $\beta = 0.15$. (Model and graphs created using MATLAB's Watts-Strogatz package[4])

2.2 Hierarchical Clustering to Model Friend Circles

In order to see which students are friends with each other, we used hierarchical clustering to create friend circles based on the clustering of the nodes. To accomplish this, we created a distance matrix table that shows the shortest geodesic distance between any two nodes. Afterwards, the average-linkage distance metric was used to define the clusters, which calculates the average distance of all nodes between clusters. This specific metric makes sense to use because it allows for easier interpretability of the distance between clusters, showing the average distance of all nodes between two clusters.

For dividing up the data into discrete clusters, we used a cutoff value of 2 for the average distance between clusters. Using this method and cutoff value makes intuitive sense because a cutoff value of 2 indicates that the all of the nodes within a cluster have an average distance of 2 or less within the group, which is what we would expect of a friend group.

2.2.1 Friend Circle Size vs. Blocking Group Size Limit

After conducting the hierarchical clustering, we can visualize the established clusters using a dendrogram that illustrates the clustering patterns in the network. For the purposes of easy visualization, we used a smaller subset of $N = 50$ since visualizing the tree would be too difficult with too many nodes (see Figure 2).

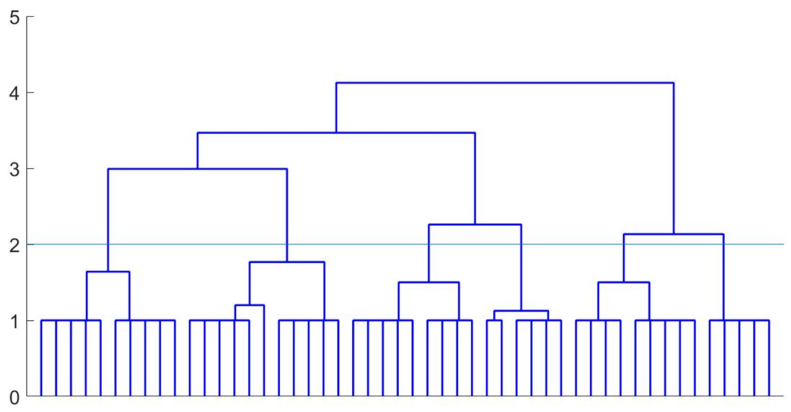

Figure 2: Dendrogram illustrating the clustering pattern of a Watts-Strogatz Social Network (N=50), with a cutoff value of 2

From looking at this graph, we see that the majority of the clusters exceeds the 8-person blocking group limit, with cluster sizes ranging from 5 to 11 and only 2 of the 6 clusters actually falling within the 8-person limit. This indicates that most of the established friend circles will be forced to experience some form of splitting, which will lead to inevitable drama and social tension.

Our results indicate that it would be worthwhile to try modeling the housing process using different size caps for blocking groups to see if there is a more optimal limit that would minimize the separation of friends during blocking.

[2]http://epubs.surrey.ac.uk/2709/1/Hamill_Gilbert11.pdf
[3]http://www.scholarpedia.org/article/Small-world_network#Watts.E2.80.93Strogatz_networks
[4]http://www.mathworks.com/help/matlab/math/build-watts-strogatz-small-world-graph-model.html

Because there are many clusters greater than 8, we predict that it would be better to have the blocking size limit be larger.

2.3 Modeling Blocking Group Formation and House Sorting

When modeling blocking group formation and house sorting, rather than trying to model the creation of both blocking and linking groups, we decided to simplify the model by viewing two blocking groups that are linked together as one large blocking group, and sorting these "large blocking groups" into one of the four neighborhoods rather than to individual houses. This is a valid simplification because the houses are organized into neighborhoods based on proximity, so this should not have a significant effect on our distance calculation later.

2.3.1 Two Methods for Blocking Formation

To model blocking group formation, we utilized two different methods and later compared their results:

1. Compare the cluster size with the blocking limit. If the cluster size exceeds the limit, create as many groups with the blocking size limit as possible, and then puts any leftovers into a final group (e.g. a cluster of 13 with a blocking group size limit of 5 would lead to groups of 5, 5, and 3)
2. Compare the cluster size with the blocking limit. If the cluster size exceeds the limit, divide the group up as evenly as possible while abiding by the limit (e.g. same example with this method leads to groups of 4, 4, and 5).

For both cases, if the size is smaller than the blocking limit, then everyone in the cluster is placed into the same blocking group.

2.3.2 Modeling House Sorting

After creating blocking groups, we randomly assigned each blocking group to one of the four housing neighborhoods, ensuring that the number of students in each neighborhood did not exceed its accomodation constraint. The values for these constraints were set to be 30 for each neighborhood because from checking the Harvard College Student Directory, each of the four neighborhoods are approximately the same size.[5] Additionally, we needed to add a buffer space in order to ensure that all the houses get properly sorted (rather than using $N/4 = 21.25$), since the varying blocking group sizes determine how the students will be divided between the neighborhoods.

2.4 Modeling Friendship Distance

In order to calculate geographic distances between neighborhoods, we assigned the following numbers to the four neighborhoods: 1, 2, 3, and 6. These indicate a very general position of where the neighborhoods are located relative to each other. The reason that the fourth neighborhood was assigned to be 6 is because it is the most far removed neighborhood from the other 3, while the other 3 are relatively close to one another. The distance between two neighborhoods is then calculated by subtracting their corresponding values from each other and taking its absolute value.

Afterwards, we created two different metrics to calculate how close students were to their friends after sorting. The first method was simply calculating the average distance of a student's immediate connections:

$$d_i = \frac{\sum_{j|n_{ij}=1} g_{ij}}{\sum_{j|n_{ij}=1} 1}$$

where n_{ij} is the network distance between i and j (based on the Watts-Strogatz model).

The second metric can be written out as:

$$d_i = \sum_{j \neq i} \frac{g_{ij}}{n_{ij}}$$

where g_{ij} is the geographic distance between i and j (based on neighborhood distance).

The reasoning behind this metric is so that information about every single node can be used to calculate the distance, rather than just the immediate neighbors. By weighting the geographic distance using the network distance, we can place more importance on closeness of immediate friends and less importance on more distant friends.

The calculation of these distance metrics is the reason that we had to use such a small N value. Because this calculation involves calculating the distance of each student from every other student, the complexity is $O(N^N)$, significantly limiting our ability to model larger populations.

[5]`http://facebook.college.harvard.edu/`

2.5 Simulations

After creating our models, we ran simulations to determine how our friend metric changes as we adjust the blocking group limit. For the simulations, we tested limit values from 1 to 20 (so in our model, that would be even numbers from 2 to 40, since we assume two linking groups form one large blocking group). We chose to not exceed the value of 20 because beyond that, it would be infeasible for students to create groups since it would be logistically too difficult to organize the groups.

For each limit value, we then ran 40 simulations where we created the social network, blocking groups, and housing assignments, and then we calculated the average and standard deviation of the 40 calculated metrics.

3 Analysis

3.1 Results from Using Blocking Method 1

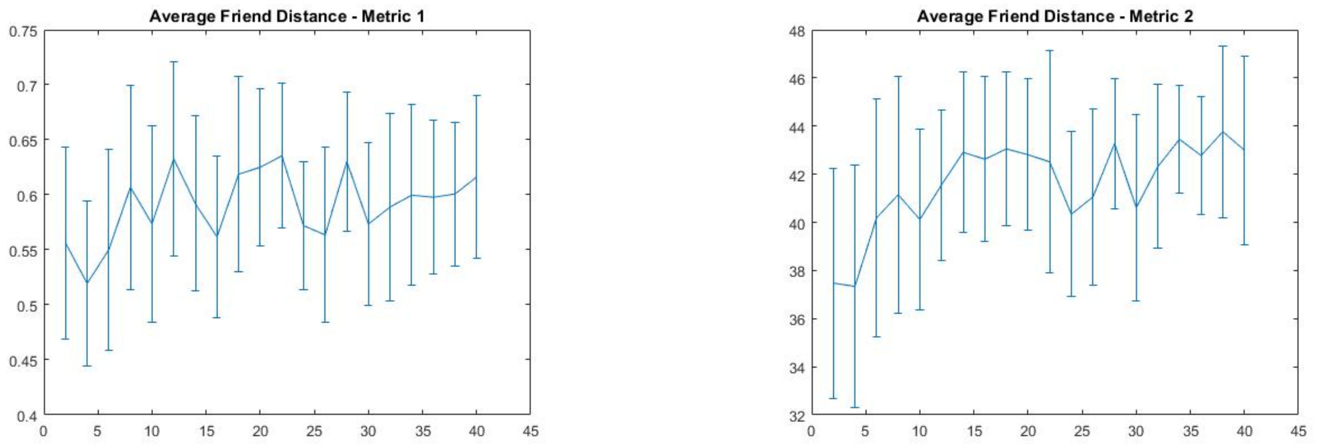

Figure 3: Change in average friend distance as a function of blocking size limit, using blocking method 1

From Figure 3, we can see that as we increase the blocking size limit, the average friend distance increases for both metrics. These results are very surprising because this is counterintuitive to what we had originally anticipated. Because most of the friend clusters were greater than 8 people, we expected the average friend distance to decrease as the blocking group size increased, since this would allow for more friends to stay together in the same house.

A possible reason for this could be due to the blocking method. With this blocking method, it is very easy to see that the splitting of clusters that exceed the blocking limit will oftentimes lead to unbalanced blocking groups. For example, if there is a cluster of 9 students but the blocking limit is 8, then the two blocking groups that would be created would be of size 8 and 1. As a result, it is completely plausible that the data is actually skewed by the few number of people who end up being isolated from the rest of their friend group, since they likely will end up being much further removed from their friends.

To determine whether this is actually the case, we tried using blocking method 2, which salvages this problem by splitting up clusters evenly.

3.2 Results from Using Blocking Method 2

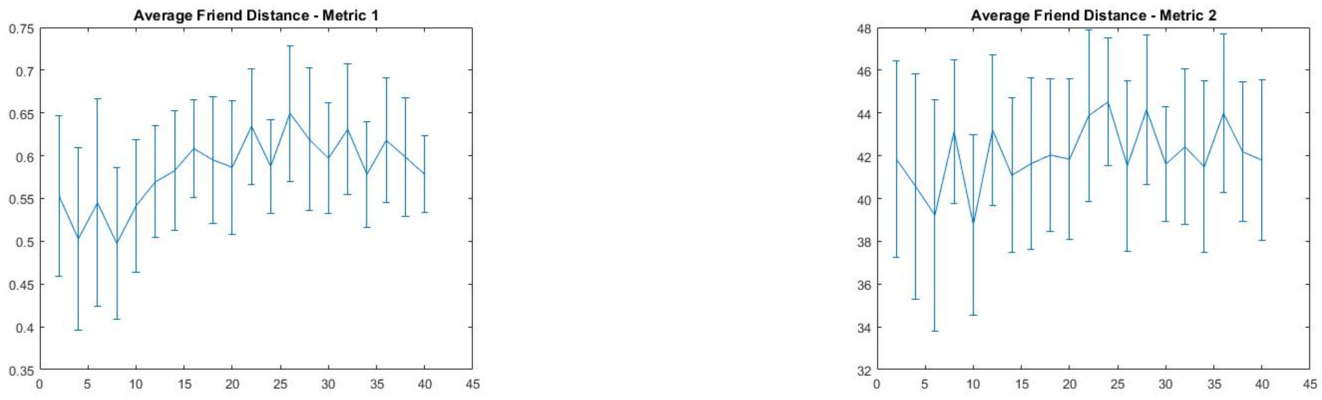

Figure 4: Change in average friend distance as a function of blocking size limit, using blocking method 2

From Figure 4, we can see that although it is not as apparent as in the first method, there still seems to be a slight increase in the average friend distance as the blocking limit increases. Because this is still the case after using blocking method 2, we could therefore infer that this positive correlation is not due to the unbalanced splitting of clusters when determining blocking groups.

3.3 Histograms of Average Distances for Individuals

For Figures 3 and 4, we see a very high standard error surrounding the mean at each point. To get further insight as to why we see our counterintuitive results, it might be useful to observe the distribution of the friend distances for every individual in order to see what could be skewing our data. For this exploration, we will look at the blocking limit that resulted in the highest value using the blocking method 2 and metric 2, which is 24 in this case.

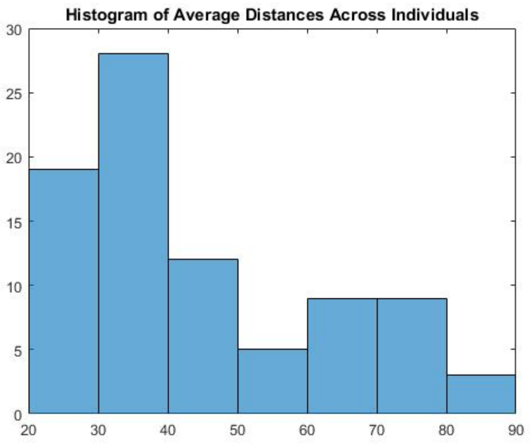

Figure 5: Change in average friend distance as a function of blocking size limit, using blocking method 2

Based on this histogram (Figure 5), it is evident that the distribution of the average distance for individuals is right skewed. Most of the data points seem to be lower, but there are some outliers that have higher average distances that are likely contributing to the heavy skewing of the data.

As a result, it is plausible that the amount of skewing increases as the size of the blocking limit increases as well. This makes sense because for as the limit value decreases, the sorting algorithm approaches a simple random sort, where each individual is sorted completely randomly into one of the four neighborhoods. With more randomness, it makes sense for the data to be less biased, whereas larger blocking groups could result in more bias in distance calculation.

Additionally, the number of individuals is not very large, so it is also very plausible that this distribution arose simply due to the small sample size. To be much more confident about the distribution, it is necessary to have a much larger sample.

4 Discussion

Based on the results produced from our simulations, we should recommend that Harvard actually decrease the blocking size limit in order to maximize the proximity of friends for every student. However, these findings only apply to this specific model, so it is important to assess the validity of this model before making any definitive conclusions.

4.1 Assumptions

One aspect of this model that makes it very difficult to deal with is its hierarchical nature. Because many of its sub-models are contingent on the assumptions of other sub-models on which they are built, it is easy to see how the whole model could become flawed if a small assumption on a lower-level model is incorrect. Here we will exam the biggest assumptions we made in creating this model.

4.1.1 Watts-Strogatz Model

One very big assumption on which everything else rests upon in this model is that the Harvard social networks could be modeled using the Watts-Strogatz Model. Although we discussed the benefits of using this model, it does have its setbacks as well. Three hallmark characteristics of social networks – communities, nodes with high degree of connectivity, and positive assortativity – are not present in the Watts-Strogatz Model, which decreases its modeling power for simulating social networks.[6]

Additionally, further research could be done on finding the best parameters to utilize for the Watts-Strogatz Model. Although there has been research done on the maximum number of acquaintances one could have[7], there has not been

[6] `http://epubs.surrey.ac.uk/2709/1/Hamill_Gilbert11.pdf`
[7] `havehttp://onlinelibrary.wiley.com/doi/10.1111/j.1548-1433.2011.01369.x/abstract`

significant research conducted about the number of close friends one has. By doing survey research on the Harvard population specifically, it might be possible to find a better parameter estimate for K. Furthermore, it might be useful to conduct a sensitivity analysis on the β value to see which value leads to a more accurate portrayal of social networks in Harvard.

4.1.2 Hierarchical Clustering

The process of clustering also involved making many assumptions that could be validated at a later point. The first big assumption is using the average linkage metric for creating clusters. Although it makes intuitive sense to use this measure, there are other metrics that could be used as well, such as complete linkage and weighted average linkage. It would be interesting to explore how utilizing these different methods affects the overall results of our simulations.

Another assumption we made was setting the cutoff average distance at 2 when creating discrete clusters. In addition to setting the value for the cutoff, we also had to make a choice as to the measure we would use for defining the cutoff. For this model, we utilized the average distance between clusters as our primary metric. However, we also had the option to use inconsistency coefficients to define clusters as well. Because this provides information about how the height of a link compares to the others below it, this could have potentially affected our results as well and cluster the nodes differently from our method.

4.1.3 Average Friend Distance Calculation

Another assumption we made was determining which metric to use to calculate the average friend distance. For this specific model, we utilized the average distance from immediate neighbors as well as the total distance from all students, weighted by how close of a friend the student is. However, there are many other metrics that could have been used in order to make this calculation, since this is a metric that was not clearly defined before conducting this project. Specifically, it would be interesting to observe how the results change if we increased or decreased the weighting of the friends distance metric, for example, using $\frac{g_{ij}}{n_{ij}^2}$ or $\frac{g_{ij}}{n_{ij}^3}$ in the summation to place more emphasis on close friends in the calculation.

4.2 Limitations

The biggest limitation of this project was the computational cost of calculating the average friend metric. Because of the overwhelmingly large amount of memory and time it took to run the code, we were forced to limit our sample size to 85 and our number of simulations to 40. This is an incredibly small fraction of the actual undergraduate population at Harvard (approximately 5000 non-freshmen undergrads), which is a substantial shortcoming for our model. Additionally, being unable to run many simulations also made our data much more skewed and noisy. If we had been able to implement a much more efficient piece of code, this could have allowed us to sample a much larger population and conduct many more simulations, which would have made the results of our data much more accurate.

4.3 Further Development

There are various directions we could take for further developing this project. One direction is trying to actually model the creation of both blocking and linking groups and sorting them into houses, rather than sorting one large blocking group into a neighborhood. This would be an interesting development because even if students end up in the same neighborhood, it makes intuitive sense that students in the same house will be much closer than students who are in the same neighborhood or different house. If linking groups were modeled, it would also be interesting to try modeling the housing system if the number of linking groups increased to 2.

Another interesting direction would be to see if there are any alternative options for creating housing assignments that are completely different from the blocking system. One option could be to have every single student rank their top houses and then create an algorithm that optimizes the frequency of top choices. Afterwards, the two housing systems could be compared to determine which one is more effective at ensuring friends stay together.

5 Conclusion

By utilizing various modeling techniques, we were able to create a model of the Harvard housing system, which indicated that it is better to have smaller blocking groups rather than larger ones to maintain close proximity of friends. However, there were many assumptions nested within the model which could use further exploration, and it is likely that these counterintuitive results are a consequence of these limitations.

6 References

- Hamill, L. & Gilbert, N. Centre for Research in Social Simulation, University of Surrey. Simulating Large Social Networks In Agent-Based Models: A Social Circle Model.
- Harvard University. Harvard College Student Directory.

- MathWorks. Build Watts-Strogatz Small World Graph Model.
- Porter, M.A. Oxford Centre for Industrial and Applied Mathematics, Oxford University. Small-world network.
- Ruiter et al. Dunbar's Number: Group Size and Brain Physiology in Humans Reexamined.
- Watts, D.J. & Strogatz, S.H. Department of Theoretical and Applied Mechanics, Cornell University. Collective dynamics of 'small-world' networks.

7 Appendix: MATLAB Code

7.1 Creating the Watts-Strogatz Model (From MATLAB Documentation)

```
% Copyright 2015 The MathWorks, Inc.

function h = WattsStrogatz(N,K,beta)
% H = WattsStrogatz(N,K,beta) returns a Watts-Strogatz model graph with N
% nodes, N*K edges, mean node degree 2*K, and rewiring probability beta.
%
% beta = 0 is a ring lattice, and beta = 1 is a random graph.

% Connect each node to its K next and previous neighbors. This constructs
% indices for a ring lattice.
s = repelem((1:N)',1,K);
t = s + repmat(1:K,N,1);
t = mod(t-1,N)+1;

% Rewire the target node of each edge with probability beta
for source=1:N
    switchEdge = rand(K, 1) < beta;

    newTargets = rand(N, 1);
    newTargets(source) = 0;
    newTargets(s(t==source)) = 0;
    newTargets(t(source, ~switchEdge)) = 0;

    [~, ind] = sort(newTargets, 'descend');
    t(source, switchEdge) = ind(1:nnz(switchEdge));
end

h = graph(s,t);
end
```

7.2 Hierarchical Clustering

```
% create distance matrix between all nodes in a Watts-Strogatz Network
dist = distances(WattsStrogatz(50,4,0.01));
dist_agree_vector = squareform(dist);

% clustering based on average metric
dist_agree_clustering = linkage(dist_agree_vector,'average');

% create dendrogram of network
figure(3)
h3 = dendrogram(dist_agree_clustering_a,0);
refline(0,2);
set(h3,'LineWidth',2)
set(gca,'FontSize',20,'XTickLabel','')
axis([-inf inf 0 5])

% divide nodes up into clusters with average distance cutoff of 2
clusters = cluster(dist_agree_clustering,'cutoff',2,'criterion','distance');
```

7.3 Cluster Size

```
function size_vector = ClusterSize(vector)
% find number of uniqe clusters
num_clusters = numel(unique(vector));
size_vector = zeros(num_clusters,1);
for x = 1:num_clusters
    % find number of elements in each cluster
    size_vector(x) = sum(vector==x);
end
end
```

7.4 Blocking Method 1

```
% parameters: clusters - 1D vector with cluster assignment for each node
%             limit - blocking group size limit
% output: blocking_groups - 1D vector with blocking group assignment for
% each node

function blocking_groups = create_blocking_groups(clusters,limit)
% find number of unique clusters
num_clusters = numel(unique(clusters));
sizes = ClusterSize(clusters);
blocking_groups = zeros(size(clusters));

% number blocking groups from 1 to # of blocking groups
group_num = 1;
for x = 1:num_clusters
    % find all indices of nodes in a specific cluster
    indices = find(clusters==x);
    % if the cluster size is less than limit, add all elements in cluster
    % to one blocking group
    if sizes(x) <= limit
        for y = 1:length(indices)
            i = indices(y);
            blocking_groups(i) = group_num;
        end
        group_num = group_num + 1;
    % if cluster size is greater than limit, keep creating blocking groups
    % of size limit until cluster is empty
    else
        cont = true;
        while cont
            if limit <= length(indices)
                % randomly sample 'limit' number from indices
                block = datasample(indices,limit,'Replace',false);
                % remove the sampled block so there is no double count
                indices = setxor(indices,block);
                for y = 1:length(block)
                    i = block(y);
                    blocking_groups(i) = group_num;
                end
                group_num = group_num + 1;
                % if there are no more elements in the cluster, break out
                % of loop
            elseif isempty(indices)
                cont = false;
            else
                for y = 1:length(indices)
                    i = indices(y);
                    blocking_groups(i) = group_num;
                end
                group_num = group_num + 1;
                cont = false;
            end
        end
    end
```

```
end
```

7.5 Blocking Method 2

```
% parameters: clusters - 1D vector with cluster assignment for each node
%             limit - blocking group size limit
% output: blocking_groups - 1D vector with blocking group assignment for
% each node

% creates blocking groups
function blocking_groups = create_blocking_groups2(clusters,limit)
num_clusters = numel(unique(clusters));
blocking_groups = zeros(size(clusters));
group_num = 1;
for x = 1:num_clusters
    indices = find(clusters==x);
    splits = split_cluster(limit,length(indices));
    for y = 1:length(splits)
        size_vector = splits(y);
        block = datasample(indices,size_vector,'Replace',false);
        indices = setxor(indices,block);
        for z = 1:length(block)
            i = block(z);
            blocking_groups(i) = group_num;
        end
        group_num = group_num + 1;
    end
end

% function for splitting up a cluster evenly if it exceeds the blocking
% size limit
function groupsizevector = split_cluster(limit,cluster_size)
num_groups = ceil(cluster_size/limit);
groupsizevector = zeros(num_groups,1);
k = floor(cluster_size/num_groups);
num_k_plus_one_groups = mod(cluster_size,num_groups);
for x = 1:num_k_plus_one_groups
    groupsizevector(x) = k+1;
end
for x = (num_k_plus_one_groups+1):num_groups
    groupsizevector(x) = k;
end
```

7.6 Creating Housing (Neighborhood) Assignments

```
% parameters: limits - blocking size limit
%             clusters - 1D vector of blocking assignments for each node
% output: sort_vector - 1D vector with neighborhood assignments for each
% blocking group

function sort_vector = sort_neighborhoods(limits,clusters)
clustersizes = ClusterSize(clusters);
num_clusters = length(clustersizes);
sort_vector = zeros(num_clusters,1);
for x = 1:num_clusters
    % cont variable is so loop continues until housing assignment is
    % completed or fails
    cont = true;
    % it's possible for the random housing assignment to not work. If
    % that's the case, then break from loop. This will only happen if a
    % blocking group cannot be added to any of the four neighborhoods due
    % to the constraint
    vals = [false false false false];
    while cont
        sort = randi([1 4],1,1);
        % called when sorting fails
        if sum(vals)==4
            error('sorting failed');
        else
            if clustersizes(x)<=limits(sort)
                % sorting into the river houses
                if sort ~= 4
                    sort_vector(x) = sort;
                    limits(sort) = limits(sort)-clustersizes(x);
                    cont = false;
                else
                % sorting into the Quad houses
                    sort_vector(x) = 6;
                    limits(sort) = limits(sort)-clustersizes(x);
                    cont = false;
                end
            else
                vals(sort) = true;
            end
        end
    end
end
```

7.7 Calculating Average Friend Distance Metric 1

```
% parameters: distances - nxn matrix with all distances between all
% possible pairs of nodes
%             sorted - list of neighborhood assignments for blocking group
%             clusters - list of friend cluster assignments for all nodes
% output: mean_distance - average friend distance value
%         distance_vector - 1D vector containing all the distances from
%         every node

function [mean_distance,distance_vector] = friend_distance(distances,sorted,clusters)
num_students = length(clusters);
distance_vector = zeros(num_students,1);
for x = 1:num_students
    distance_sum = 0;
    % neighborhood is the current neighborhood for student x
    neighborhood = sorted(clusters(x));
    num_neighbors = 0;
    for y = 1:num_students
        % only calculate average for immediate neighbors
        if distances(y,x)== 1
            distance_sum = distance_sum + abs(neighborhood - sorted(clusters(y)));
            num_neighbors = num_neighbors + 1;
        end
    end
    distance_vector(x) = distance_sum/num_neighbors;
end
mean_distance = mean(distance_vector);
end
```

7.8 Creating Average Friend Distance Metric 2

```
% parameters: distances - nxn matrix with all distances between all
% possible pairs of nodes
%             sorted - list of neighborhood assignments for blocking group
%             clusters - list of friend cluster assignments for all nodes
% output: mean_distance - average friend distance value
%         distance_vector - 1D vector containing all the distances from
%         every node

function [mean_distance,distance_vector] = friend_distance2(distances,sorted,clusters)
num_students = length(clusters);
```

```
distance_vector = zeros(num_students,1);
for x = 1:num_students
    distance_sum = 0;
    neighborhood = sorted(clusters(x));
    for y = 1:num_students
        % don't include distance to oneself
        if distances(y,x)~=0
            distance_sum = distance_sum + abs(neighborhood - sorted(clusters(y)))/(distances(y
        end
    end
    distance_vector(x) = distance_sum;
end
mean_distance = mean(distance_vector);
end
```

7.9 Simulation Code

```
% parameters: sizelimit - blocking group size limit
% output: average - average friend distance for simulation
%         distance_vector - vector of all friend distances for each node

function [average,distance_vector] = simulations(sizelimit)
dist = distances(WattsStrogatz(85,4,0.01));
dist_agree_vector = squareform(dist);
dist_agree_clustering_a = linkage(dist_agree_vector,'average');
clusters = cluster(dist_agree_clustering_a,'cutoff',2,'criterion','distance');

blocking_groups = create_blocking_groups2(clusters,sizelimit);
neighborhoods = sort_neighborhoods([30 30 30 30],blocking_groups);
[average, distance_vector] = friend_distance2(dist,neighborhoods,clusters);
end

% code for many housing simulations for different blocking limits. Creates
% plot of results of mean distances with error bars

function run_simulations()
avg_distances = zeros(20,1);
std_dev = zeros(20,1);
for x = 1:20
    runs = zeros(30,1);
    for y = 1:40
        [average, ~] = simulations(2*x);
        runs(y) = average;
    end
    avg_distances(x) = mean(runs);
    std_dev(x) = std(runs)/2;
end
errorbar(2:2:40,avg_distances,std_dev)
title('Average Friend Distance - Metric 2');
end
```