Simulating The Propensity to Cheat in Classroom Settings

Introduction

In a perfect world, nobody would ever cheat. Sadly, this is not the case and cheaters are very prevalent in education. Furthermore, one of the most common ways to cheat is plagiarism. A lot of effort has been made to detect plagiarism. Specific programs even exist to help graders detect plagiarism of source code. Such programs are actually quite effective, even when students try to make small changes or additions to not get flagged. Xin Liu, who has implemented such a system, stated, “The source code plagiarism detection system…has checked over 4,000 jobs. It's false alarm rate (0.0) and omission rate(0.05)” [1]. So if cheaters are able to be found so effectively, why the need for further research?

It is easy to find two homework submissions that are identical, but there is more to be discovered? This paper aims to dive deeper into the facets of cheating. What if it could be decided who was the sender and who was the receiver? What if one homework submission was spread about the class like a disease and the original source of cheating could be detected? Imagine predicting and preventing environments that will be conducive to cheating. These are the questions that will drive this paper.

In order to model this problem, a class of students will be viewed as a network of multi-agent systems (MAS) linked by a “friendship” and decision-making of whether to share their homework with friends or not. Both the decision to be friends and the decision to cheat will be simulated in a MAS manner. This paper will discuss the “flow” of cheating and if it is possible to predict such flow from analyzing the results of our simulation..

Previous work

Assigning responsibility to agents in a MAS is a topic discussed in depth in literature. Yazdanpanah et al. offer definitions of responsibility in MAS and propose methods for assigning responsibility in [2]. They assert that an agent or group of agents is responsible for an outcome if they “had a strategy to preclude [the outcome] given their knowledge” [2, p. 593]. They further specify that a group of agents is responsible for an outcome if and only if all agents participate in the action that resulted in the outcome.

They characterize mechanisms for assigning responsibility as *forward* when the mechanism predictively assigns responsibility before agents make choices and *backward* when the mechanism assigns responsibility based on a history of agents' choices. Forward responsibility is computed when the behavior of a MAS needs to be predicted, and backward responsibility is computed when the behavior of a MAS needs to be analyzed. Such backward analysis techniques provide valuable information about the characteristics, motives, and intention of agents in a MAS.

Backward responsibility is utilized extensively in covert networks. Covert networks are social networks whose vertices represent criminals, terrorists, or other individuals with malicious intents. The edges in covert networks represent relationships between criminals, such as partnerships in crime. Criminals in the network attempt to behave in ways that relinquish them from the responsibility of their actions.

Dey and Medya explain in [3] that to construct a graph that models a covert network, each vertex is assigned a score. The score reflects the relative influence or contribution of the vertex to the current status and is known as the vertex's centrality. When assigning backward responsibility to vertices in a graph, the vertices' centralities are assigned using various algorithms, such as core centrality. Dey and Medya discuss the use of core centrality, which is computed as a function of a vertex's degree and the degrees' of its neighbors. Core centrality method is useful for assigning backward responsibility in covert networks.

There are many ways to measure centrality which could prove useful for this research. Two additional centrality methods are eigenvector centrality and percolation centrality. Eigenvector centrality is designed to measure the influence of a node inside a network based on connections to "high-quality" nodes [4]. This will be helpful in finding students who have many cheat-prone friends. The other method is percolation centrality. Percolation centrality handles the scenario where some form of virus or condition spreads through the graph. The percolation centrality assigns each node a value based on its likelihood to continue the percolation of such condition [5]. These two metrics combined can be useful in the analysis of our student-graph.

My contribution

We are proposing implementing a simulation of a social network or MAS of students. Within this network each student has the option of deciding who their friends will be and whether or not they will cheat. Their ability to cheat is manifested by either receiving homework from another student and/or passing homework on to another student.

First some common terminology must be understood for conversation to continue. The class we are simulating is a computer science course. Cheating is submitting another student's source code as their own. Source code is the artifact transferred between students. A cheater is a student who uses another's source as their own or sends their source code to another student. A sender is a student who sends source code to another student. A requester is someone who asks for the source code of another student and may become a cheater as well as the sender. An original cheater is a requester who is not influenced by peer pressure and begins a trail of cheating amongst the MAS.

Whether or not students cheat will depend on the social network within the MAS. If a student is a sender they will only attempt to send source code to their friends. This will allow us to view the flow of cheating among social networks. Senders will only send upon request. First the metrics must be verified by the requester and then the requester will ask from each of their friends. This friend's metrics will then be checked to see if they will send their source to the requester/receiver.

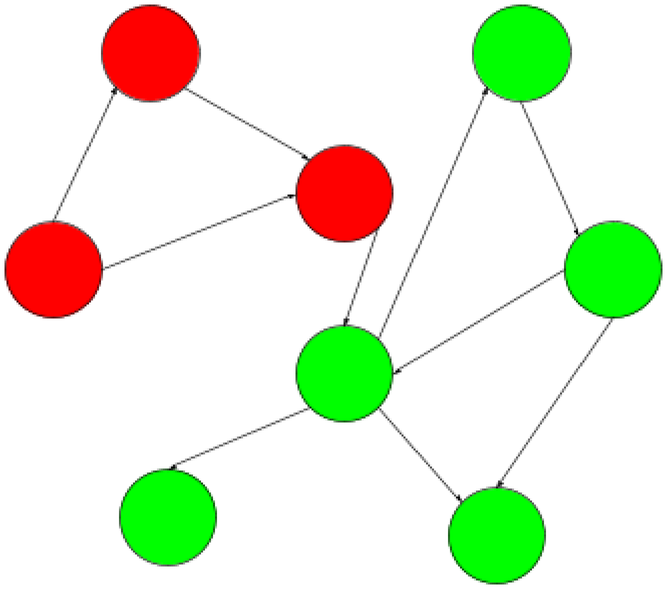
Figure 1: A social network of students

Figure 1 shows a representation of the social network of students. The green dots represent students who have finished and did not cheat and the red dots represent students who cheated to complete their assignment. Although we do not know who was the sender and who were the receivers in this social network; merely because we are looking at the results of a simulation here and not watching it in real time. It is observable that there are 2 subgraphs within this graph; the cheaters and the non-cheaters. Another possible scenario could be where there are multiple subgraphs of cheaters. This would indicate that there are multiple original cheaters who requested help from their friends who then became senders. This allows us to visually see the flow of cheating from friends to friends and how this can end the flow at certain nodes when some friends are not willing to send source code.

The results will be analyzing the impact of various metrics in the flow of cheating in the MAS of students. These metrics include but are not limited to: number of friends, moral level, procrastination level, peer pressure level, and desperation level. By varying these metrics we can see their impact on the flow of cheating by visualizing graphs similar to the figure shown above. The goal is to view the interaction of a MAS that behaves according to multiple factors. Similarly to linear regression we will be able to see which factors have the greatest impact and which of those do not.

Experimental Design

The entire implementation is divided into 5 modules. These are named main, configuration, visual, configuration, student and class. This portion of the report will have sections discussing each of the modules with commentary on their interactions.

Main

The main module is a fairly simple one. It consists of importing the other modules and creating a class object. This object then interacts with the visual module to display the simulation in real time. The for loop is the driver which steps us through each day of the simulation.

Configuration

The configuration module is algorithmically simple but syntactically essential. The configurations are essentially global constants that have more meaning. In order to give the syntactic power to allow us to see the meaning of these constants they are defined in the configuration module and used throughout the other modules.

The module is broken into sections by comments which heightens the organization of this module. They are organized according to where the constants are used within the other modules. The demographic configuration consists of a max cheat level which is used to calculate other cheating metrics upon which students make their decisions.

The other sections have a number of personalities, minimum and maximum friends, which come into play when building the multi-agent system and their connections. The tolerances in these sections are what determine the agents' decisions of who their friends are. The settings can also be changed to determine how many days the simulation will run and the size of the class. The minimum moral level for sending and requesting homework in the simulation is a percentage of the maximum cheat level which can be set in the configuration file as well as the percentage of desperation and the peer pressure level can affect a student's moral level. Before passing control to the main module, the configuration module validates that all configurations are compatible with each other.

Because it is not realistic for the distribution of cheaters and procrastinators to be evenly distributed across a range, the user configures these distributions through lists of floating point numbers in the range [0, 1]. For each index *j* in these lists, the number at index *j* indicates the proportion of students in the class with a cheating or procrastination level of *j*. The lists of cheating and distribution levels are computed and then randomly assigned to each student.

In summary, the following configurations are used throughout all other modules in the simulation:

- C_DST: **c**heat **dist**ribution - the distribution of cheating levels, expressed a list of fractional numbers.
- P_DST: **p**rocrastination **dist**ribution - the distribution of procrastination levels, expressed as a list of fractional numbers.
- NB_ST: **n**umber of **s**tudents in the simulation
- TH_RQ: **th**reshold for **req**uesting - a number on the range [0, 1] directly proportional to the minimum amount of cheat units required for a student to make a request
- TH_SD: **th**reshold for **send**ing - a number on the range [0, 1] directly proportional to the minimum amount of cheat units required for a student to send their homework to another student
- DSP_LV: **desp**eration **lev**el - a number on the range [0, 1] directly proportional to the students propensity to cheat because of their desperation (explain desperation before)
- PP_LV: **p**eer **p**ressure level - a number on the range [0, 1] directly proportional to the student's propensity to cheat because of peer pressure

Student

The student module contains two very important portions of code that make this a multi-agent system. They are the cheating and friendship portions of the implementation. Each student has a unique identification number, a cheat level which can be considered their moral level, a procrastination level, a peer pressure level, and a desperation level. The cheat and procrastination levels are static characteristics determined according to the user-defined distributions. The peer pressure and desperation levels are computed as functions of other properties of the student, which are discussed in greater detail later. All levels are defined such that the higher the level of property *i*, the more inclined the student is to engage in activity *i.* For example, a higher procrastination level indicates a greater likelihood to procrastinate.

Each student also has a progress variable which tracks their completed progress on an assignment. There is a universal due date shared by students. They also have two boolean flags for whether or not the student is finished and if the student has cheated. Each student is then assigned a calculated start day based on their procrastination level. Their finish day is randomly selected between the start day and the last day of the assignment. Then, the work per day needed to be done by each student in order to make these start and finish days succeed is calculated. The work per day assumes that a student does an even amount of work each day in the period of time between their start and finish day.

The student is assigned a personality as a random integer between 0 and a user-configured amount of personalities. They are also assigned a preferred number of friends as a random integer between two user-configured amounts. Each students' friends are determined using a multi-agent approach. A student has a function that selects one friend at a time from among all other students in the class. During initialization, this function is called on all students whose amount of friends is less than their preferred amount of friends until all students have met their preferred amount of friends.

To select a friend from the class, a student identifies the friends of their friends with whom they are most compatible. Two students increase in compatibility as the difference between their cheating and procrastination levels decreases and if their personality types are the same. If the amount of a student's friends' friends is less than a user-defined amount, then the student randomly selects enough students to meet the amount and then picks the most compatible student from among the candidates. When student *i* makes friends with student *j*, student *j* makes friends with student *i.* By first selecting a friend from the friends of a student's friends, the system models the real-life experience of getting to know people based on mutual relationships.

The student's use day method is called by each student when the class use day method is called to run the simulation. This method increments the progress on their assignment, checks if they have finished the assignment, and decides if they will cheat.

If the student decides to cheat, the student requests homework from their friends. This is according to the following metrics: they can't be finished, they can't have previously cheated, and they must have started on the assignment, then the sum of their cheat level, desperation level, and peer pressure level must be greater than the moral cheat minimum necessary to request. Now is a good time to talk about how the desperation and peer pressure levels are calculated.

The desperation level is directly proportional to the amount of homework they have left on the assignment and inversely proportional to the amount of time they have left to complete the assignment. This proportion is then multiplied by the maximum desperation level to give us their individual desperation level.

The peer pressure level is also a proportion of a maximum peer pressure level. The proportion is the amount of friends they have that have cheated previously divided by their total number of friends.

Once a student makes a request, they go through each of their friends and the potentially send method is called on their friend. In the potentially send method, the moral level of the student must be greater than the moral cheat minimum level required, their friend must have started the assignment themselves, and they must be farther in the assignment than the requesting student. If all of these conditions are true, then the friend sends their homework. At this point both the requesting and sending students are flagged as cheaters.

Class
The final module is the class module. The class module consists of the universal due date, the current day, and a list of students. Within this module is also the code for visualizing the histograms for the distributions of cheaters and procrastinators. These graphics appear at the same time as the visual for the class simulation.

The use day method in the class module is called in the for loop in the main module. This loops through each student in the class and which then calls their individual use day method. At the end the current day is implemented. The class has some extra benefits in the report method which allows us to see what is happening through dynamic print statements based on the variables in the student class.

The implementation does not include any of the algorithms for determining who was the original cheaters in the network after the due date. This is just to create the network of students and allow the agents to make decisions of who their friends are and whether or not they will cheat. Although we wanted to make this contribution we made the decision as a goal to first get the network working and that became our end goal. This leaves for lots of room for future work in this implementation as well as modifying the metrics and methods for how students make their decisions.

Visual

The visual module enables the viewer to see the simulation graphically update in real time. The visual is based on the previously implemented turtle python module and draws a circle of black dots each representing a student. The black lines are drawn in between each student to represent a friendship connection. Throughout the simulation the dots are changed to green as the student completes their homework. The dots are given an inner red dot if they are a cheater. At the top of the visualization is the current day of the simulation. An example of the visualizations are shown in figure 2.

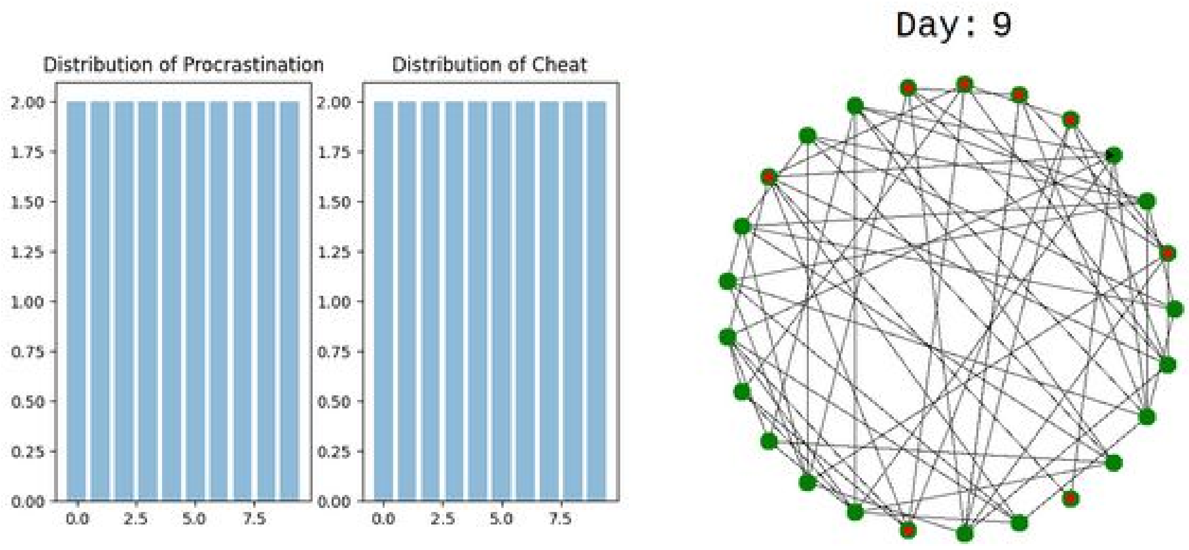

Figure 2: Sample visualizations

Results

The simulation was performed on enough sets of configurations to demonstrate the effect of each configuration on the system. Because the simulation involves a small amount of randomness, for each set of configurations, the simulation was run 100 times, and the average percentage of students that cheated was recorded. Each simulation, or sample, was given a unique ID, and a summary of each simulation with its configurations and results are summarized in Table 1. Distributions of cheating and procrastination are given a letter as a unique identifier, and Table 2 defines the distributions associated with each identifier.

Sample 1 was configured to represent the profile of a typical class. There are 50 students, few (5%) of them have a high propensity for cheating, most (65%) of them will procrastinate on the assignment, and each student has between 10 and 35 friends. The parameters for the threshold for requesting and sending homework, along with the desperation and cheat levels, were selected such that the percentage of students that cheat in the simulation hovers at a reasonable small value (13.5%). All other simulations are compared with respect to this simulation.

In sample 2, the distribution of cheaters is changed such that the propensity for cheating at the maximum level is increased from 5% to 40%. This increases the total percentage of cheaters from 13.5% to 87.6%. Such a dramatic increase in cheaters is expected. In sample 3, the distribution of procrastinators is changed such that the propensity for procrastinating at the maximum level is decreased from 65% to 1%. This increases the total percentage of cheaters from 13.5% to 14%. A larger increase in cheaters was expected, and further work is necessary to adjust the part of the code that handles a student's procrastination.

In sampe 4, the total number of students is increased from 50 to 200 students, which increases the percentage of cheaters from 13.5% to 14.1%. Although the increase is modest, this seems reasonable because as more students join the class, there are more potentially immoral students with whom any given student could be friends, as suggested in [3].

In sample 5, the threshold for requesting to cheat from another student is doubled from .2 to .4. This reduces the percentage of cheaters to 7.1%, or nearly half of the original value. Likewise, in sample 6, the threshold for sending homework to another student is doubled from .2 to .4. This reduces the percentage of cheaters to 8.6%. These reductions in cheating were expected since increasing one's threshold for cheating would decrease one's propensity to cheat.

In samples 7 and 8, the desperation and peer pressure levels are altered to encourage more cheating. In sample 7, the desperation level increases from .2 to .7, but the percentage of cheaters surprisingly drops insignificantly by .4%. This result is unexpected, and suggests that the algorithm that uses this configuration could be improved. However, in sample 8, the peer pressure level increases from .2 to .8, and the percentage of cheaters increases to 15.3%. Although the increase is modest, an increase is expected since the peer pressure level is directly proportional to a student's propensity to cheat.

In samples 9 and 10, the amount of friends is changed. In sample 9, the range over which the amount of friends per student is selected is changed from [10, 35] to [25, 45], and the percentage of cheaters increases from 13.5% to 15.8%. In sample 10, the range is changed from [10, 35] to [24, 45], and the percentage of cheaters decreases from 13.5% to 6%. Both of these results are consistent with the expectations, since having more friends opens more opportunities to send and receive homework from other people.

Table 1. Configurations and results of each simulation. Columns 2 - 8 refer to the configurations explained in section *Class* under the heading *Experimental Design.* AV_CT indicates the average percentage of cheaters in the class over 100 simulations.

ID	C_DST	P_DST	NB_ST	TH_RQ	TH_SD	DESP_LV	PP_LV	FRND	AV_CT
1	A	Z	50	.2	.2	.2	.2	[10, 35]	13.5
2	B	Z	50	.2	.2	.2	.2	[10, 35]	87.6
3	A	Y	50	.2	.2	.2	.2	[10, 35]	14.0
4	A	Z	200	.2	.2	.2	.2	[10, 35]	14.1
5	A	Z	50	.4	.2	.2	.2	[10, 35]	7.1
6	A	Z	50	.2	.4	.2	.2	[10, 35]	8.6
7	A	Z	50	.2	.2	.8	.2	[10, 35]	13.1
8	A	Z	50	.2	.2	.2	.8	[10, 35]	15.3
9	A	Z	50	.2	.2	.2	.2	[25, 45]	15.8
10	A	Z	50	.2	.2	.2	.2	[1, 10]	6.0

Table 2. Definitions of distribution

Cheating Distributions		Procrastination Distributions	
A	[.8, .1, .05, .05]	Z	[.05, .05, .25, .65]
B	[.1, .1, .4, .4]	Y	[.97, .01, .01, .01]

Future Work

A big emphasis was put on modeling the simulation after real-life logic and events. This created a slew of outcomes for each configured set of students. Each of these results seems to be realistic and feasible. The results of this simulation, however, aren't supposed to give any meaningful conclusion themselves as the simulation is configured with many arbitrarily assigned values. The point of the simulation is to provide highly-configurable and easily-produced data which can then be analyzed.

The most significant analysis of this data will involve the centrality algorithms discussed previously. The centrality algorithms are designed to predict which nodes in the graph are the most persuasive or influential. Such information could be very useful in preventing future cheating. Eigenvector centrality requires some weighted value to produce results. Our simulation is heavily based off of a cheat level (moral values), which would fit perfectly. Percolation centrality takes into account the spreading factor which is also modeled by our simulation. These algorithms are modeled by mathematical equations. A useful analysis could be found by combining these two equations or applying them sequentially in a way such that both would be taken into consideration. Such an algorithm could be designed and tested on the data produced by our simulation. Once the algorithm passes a certain accuracy on the simulation's data, the algorithm could then be tested on real-life data.

Another topic that could provide insight to educators would be the analysis of classroom rules. This would answer the question of "How do I reduce cheating." Some "rules" of the classroom could be (but not limited to):

- How many days are assignments released for
- How much chatter is permissible (friendships)
- How bad the consequences for cheating are

Being able to change such parameters in a real-life setting would be extremely expensive and slow before results came in. This simulation would be a great base for those looking to add another component: the teacher. A teacher would interact with the class in between assignments and modify the classroom rules. Some results of which parameter reduces cheating the fastest could then be extrapolated. Once again, although the original simulation doesn't produce many applicable results, it provides a great stepping stone towards meaningful results of all kinds.

References

[1] X. Liu, et al. "Plagiarism Detection Algorithm for Source Code in Computer Science Education." International Journal of Distance Education Technologies, vol. 13, no. 4, 2015. [Online]. Available: https://link-gale-com.dist.lib.usu.edu/apps/doc/A421556138/AONE?u=utah_gvrl&sid=AONE&xid=4666f3b1. [Mar. 11, 2020].

[2] V. Yazdanpanah, et al. “Strategic Responsibility Under Imperfect Information,” Agent Societies and Societal Issues, no 2, May 2019. [Online]. Available: https://dl.acm.org/doi/10.5555/3306127.3331745. [Accessed March 12, 2020].

[3] P. Dey, S. Medya, “Covert Networks: How Hard is It to Hide?,” Agent Societies and Societal Issues, no. 2, May 2019. [Online]. Available: https://dl.acm.org/doi/10.5555/3306127.3331749. [Accessed March. 10, 2020].

[4] M. E. J. Newman. "The mathematics of networks" Center for the Study of Complex Systems [Online]. Available: http://www-personal.umich.edu/~mejn/papers/palgrave.pdf. [Accessed Mar. 12, 2020].

[5] Piraveenan, Mahendra et al. “Percolation centrality: quantifying graph-theoretic impact of nodes during percolation in networks.” PloS one vol. 8 no 1, Jan 22, 2013. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3551907. [Accessed Mar. 12, 2020].