4-22-2009

Agent Communication in Multi-Agent Models of Information Cascades

Jennifer West
Trinity University

Agent Communication in Multi-Agent Models of Information Cascades

Jennifer West

A departmental thesis submitted to the

Department of Computer Science at Trinity University

in partial fulfillment of the requirements for Graduation.

April 22, 2009

Thesis Advisor

Department Chair

Associate Vice President

for

Academic Affairs

Agent Communication in Multi-Agent Models of Information Cascades

Jennifer West

Abstract

Understanding how information is transmitted and how an information cascade is formed has many applications both in understanding political and economic behavior and how to best implement economic and public policies. Many papers have shown that information cascades do occur, but they always describe a very basic situation that does not occur often in the real world. To further understand information cascades in more complex conditions, I extended a multi agent simulation model that set out to investigate information cascades in the motion picture industry. I extended the model by allowing agents to speak to other agents before making a movie viewing choice. By allowing agents to communicate, the agents were more effective in choosing high quality movies. How the information cascade presented itself altered with the agent communication. Information cascade occurred in the information the agents transmitted to one another. This resulted in the agents choices focusing their movie choices over a smaller number of movies. This result shows that given a large variety of choices where many are good choices, cascading on a single choice becomes more difficult.

Acknowledgments

I would like to thanks my thesis committee of Dr. Lewis, Dr. Watson, and Dr. Zhang for reading through the thesis and providing helpful comments. I would additionally like to thanks Dr. Lewis for helping me come up with the topic, and helping with the project. I would like to thank Dr. Watson for being tremendously helpful with any questions I had about the economic aspects of the project. Finally, I would like to thank my sister Amy for generally being an awesome sister.

Agent Communication in Multi-Agent Models of Information Cascades

Jennifer West

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introducing Information Cascades

There are many economic situations in which agents must make decisions using only incomplete information and the observed actions of others. These situations raise several questions. For one, do agents converge on a single decision? For another, how do agents receive the information? Lastly, how do agents use this information? By understanding these dynamics, economists may begin to understand phenomenon such as the housing bubble and the movements of the stock market.

An information cascade occurs when a sequence of agents each make a decision, based on an independent private signal and observations from previous agents independently of their own private information signal. The private signal provides the agent with information on the correctness of a hypothesis. In the models described in this paper, the hypothesis will always be if a certain movie is correct. In the example below the hypothesis is if a certain path is correct. The private signal has a probability, which is denoted as p, of being correct. In cases of an information cascade the agents decision, even to ignore her own private signal, is a rational choice that she decided using Bayesian updating. Bayesian updating is a statistical way to update information where evidence or observations are used to update the probability that a hypothesis is true. Bayesian updating updates the probability the hypothesis is true using the following formula:

$$P(E) = \frac{P(E|H)P(H)}{/P(E)}$$

H is the specific hypothesis, E is the new evidence, P(H) is the prior probability of H, P(E—H) is the conditional probability, P(E) is the marginal probability of E, and P(H—E) is the posterior probability of H given E.

Agents must infer the private signals of the agents who went ahead of them to make up for their own incomplete information. A common example of an information cascade is three people approaching a fork in the road. One road leads to a pot of gold and the other leads to nothing. Each person receives a personal signal about the hypothesis that going left will lead to the pot of gold. There is a 2/3 chance each person will receive a correct private signal about whether or not going left will lead to the pot of gold. Assuming the third person observes both the first and the second person choose left. If the third personal was acting rationally, she would go left even if her private signal was to go right. This example also demonstrates some problems with information cascades. Even if people are acting rationally in ignoring their private signals, they may all be wrong. In the example above, it is possible for thousands of people to choose wrong entirely because of the first two peoples decisions. The above example demonstrates a very basic example of an information cascade. Other studies show other characteristics of information cascades in more complex cases. For example, Goeree et al. [6] found that information cascades are also fragile: any new information presented to agents can end and potentially reverse a cascade.

1.2 Background

The paper that began the research into information cascade was the paper by Bikhchandani et al [3]. This paper used mathematical proofs to give insight into specific aspects of information cascade formation. The paper concluded that information cascades can explain the formation, maintenance, and change of social norm and fads. Given the paper relied only on mathematical proofs, the models can't be guaranteed to hold when applied to real people.

The classical information cascade experiment can be found in a paper by Anderson et al. [2]. In the experiment, agents make decisions sequentially and they have access to the decisions of all the agents ahead of them. They are then paid if they answer correctly. This paper assumed that the participants made decisions using Bayesian updating. Huck et al. [7] set out to find if people always made decisions using Bayesian updating. Bayesian updating is statistical inference where a person uses observations to infer the probability something being true. They discovered that roughly half of participants in their experiment made decisions using Bayesian updating. Part of the reason, they believed, for the low percentage of people using Bayesian updating to make decisions was due to their experiment being more complex than previous experiments. This added complexity may have made Bayesian updating too difficult for many of the participants in the study. Oberhammer and Stiehler [9] looked into how subjects calculated probability by requiring participants to submit maximum prices that they would be willing to pay to participate in the game. The results could not be explained by Bayesian updating or by heuristics. Also, they discovered that subjects were willing to pay more in the beginning, but as the experiment drew on they were willing to pay less and less.

Continuing to look at how confidence in previous agent's decisions progressed, Kubler and Weizsacker [8] looked into applying a positive cost to getting private signals. The paper found that participants were willing to buy more private signals in the beginning. The later participants purchased less signals, presumably because they had more faith that previous decisions were based on private data. This paper suggests that fads occur because people believe that previous decision makers made informed choices and thus the person is willing to follow the majority.

Ivo and Welch [10] investigated information cascades using experts. In this case they used investment analysts to see if being an expert made you more or less likely to follow another expert. The paper showed that the decision of a single analyst effects the decisions of the next two analysts. Also, the current prevailing consensus has a positive effect on an analysts decisions regardless of how correct it is. The prevailing consensus has a larger influence if there is a good economic condition and the promise of return is higher than usual. These results help show that people do have certain herding behavior. Herding behavior is defined as occurring when several agents make identical decisions without ignoring their private information.

To attempt to answer the question of whether information cascades exist, De Vany and Lee [5] ran a multi agent simulation of the movie industry to measure the cascade affect. They extended the previous models by having local interaction, among other changes. The author discovered that information cascades did not occur as often as expected or converge as narrowly or as quickly as the theory would suggest.

Several papers then tried to explain how certain added variables affect the information cascade. One paper by Goeree et al. [6] added longer sequences of decision and variation of how informative a signal is. The results were quite different than the standard theory. Their experiment showed fewer permanent cascades, more variation in the length of the cascades, and more alterations between correct and incorrect cascades than was previously expected. Also, they found participants were, on average, overconfident in their own private signal. Bogachan and Kariv [4] also found that subjects gave more weight to their own private signals than others. They also discovered, while trying to distinguish herding from cascades, that cascades do occur around a third of the time. The authors also looked into how perfect and imperfect information models explain information cascades. They find that the imperfect information model is best at describing occurrences such as fad and is more realistic. They also found that with imperfect information, cascades last longer and the chance of a reverse cascade drops. Alevy et al. [1] looked into the affect being a professional has on your confidence in others' decisions. They found that financial professionals, when compared to college students, better understood that other's judgments have varying degrees of quality and overall acted more in line with economic theory in that they were more consistent with using Bayesian updating to help make decisions.

1.3 Other Models

In 1992 Bikhchandani, Hirshleifer and Welch [3], who will be refered to as BHW, gave one of the earliest descriptions of information cascades. In their model, agents adopted or rejected an action based on their own private signals and the decisions of all the agents ahead of them. The agents are all standing in a line and can view the action of all the agents ahead of them. If an agent accepts the decision, the following agents can infer they received high personal signal about the action, and conversely, they can conclude an agent received a low personal signal if they rejected the action. The paper assumes agents make their decision based on Bayesian updating. If enough of the agents before a specific agent adopts the action, a rational agent may choose to accept an action even if she got a low signal because she has decreased the weight she places on her own decision.

Key results of this paper are as follow:

1. As the precision of a person's private signal increases, a correct cascade (one where agents adopt the correct choice) starts earlier and is more likely to occur.
2. A noisy enough signal can increase the probability of an incorrect cascade to as high as 0.5.
3. An agent with a high probability of choosing correctly early in the sequence can start a cascade, but a higher precision agent can shatter a cascade later on if she chooses to.
4. The precision of no cascade occurring decreases as the number of agents in the sequence increases.
5. An increase in the number of agents increases the probability a cascade will start.
6. Once started, a cascade will last forever given no new information.
7. New information can potentially shatter a long-lasting cascade.
8. As more public information is released, the correct choice becomes clearer and individuals settle into a correct information cascade.

De Vany and Lee [5] developed a multi-agent model based on the movie industry to try and to develop information cascades in this environment and compare their findings to the theoretical predictions of the BHW model. The BHW model gave only probabilities of cascades but provided no tests to give credit to its assumption, De Vany and Lee wished to fill that gap. This model found the following results.

1. An increasing value in p (the probability an agents private signal is correct) is not necessarily associated with a higher percentage of correct choices.

2. Their model found that the probability of a cascade is not nearly 1, in fact for low values of p cascades rarely occur and when they do they are very fragile.

3. Higher value of p are associated with a heavier-tailed distribution of agents over movies (as indicated by a lower α in a Pareto distribution.). This means as p increase cascades become less fragile.

Research described by De Vany and Lee [5] show that the market shares of the motion picture industry are well modeled by the Pareto-Levy distribution. This distribution is a heavy tailed distribution which means that it has more probability mass on extreme outcomes than a normal distribution. The figure below shows the effect of changing the α value:

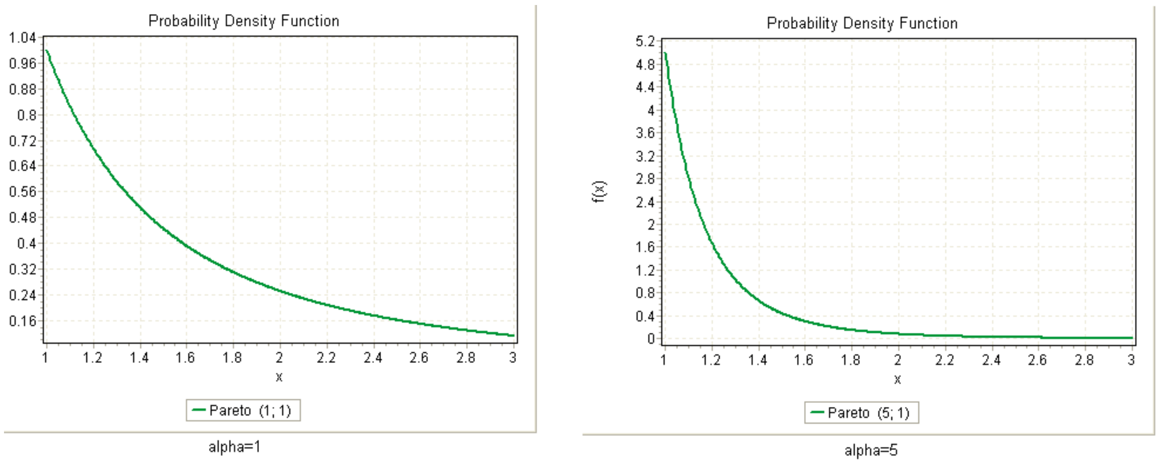

Figure 1.1: Effect of Alpha on the Probability Density Function

When related to movie market shares, a lower α value means a few movies are capturing a greater percentage of market shares when compared to higher α values.

My intention was to replicate the results from De Vany and Lee, and then add another element. Unfortunately I could not replicate the exact results De Vany and Lee found. I could not verify their first assumption; in fact I found that an increasing value of p is always associated with higher percentage of correct choices. This discrepancy between my results and theirs may be a result of a misunderstanding of what exactly their method was. Either way, I created a model, based off of theirs, and then added the ability of agents to speak to other agents and use this data in their final decision making process. My approach is described in the next chapter and the results are in the following chapters.

1.4 Motivation

Many papers on information cascades set out to determine if they occur and if they are fragile, meaning the cascade is highly prone to ending. De Vany and Lee [5] considered a third question: what types of choice distributions do information cascades produce. They hoped to understand the properties of converging information cascades statistical distributions. The answer to this question would also explain whether information cascades occur and whether or not they are fragile. De Vany and Lee choose to model movie attendance to test the choice distribution for an information cascade. Movies are a good choice for several reasons, but the most important is that they offer an opportunity for both public and private information. Information from the media that shows the market share of each movie provides an aggregate signal required for an information cascade. People also have a private idea of whether a movie will be good or bad, the personal signal aspect of the model. De Vany and Lee expanded on the basic model to make it more realistic. They allow learning by letting agents gather information about movie quality from others. They also allow local interactions among the agents; each can personally interact with the agent directly ahead of them in line.

There are a few problems I found with the model proposed by De Vany and Lee. The first and biggest problem I found, which will not be solved in this paper but will be improved upon, is that the model is not a realistic representation of how people make movie decisions. In the real world, people do not go to the box office without knowing anything about a movie and then use only market shares and information from the person in front of them. That said, making a model completely realistic would be impossible, but adding more realistic elements to the model would give a better idea about whether an information cascade can occur. In the motion picture industry, advertisers for a specific movie hope to create an information cascade of sorts to increase their revenue. Whether such a thing is even possible would help determine the best advertising method.

There are a few specific issues with De Vany and Lees method that I will address in this paper. One problem with previous models of information cascades concerns agent interaction. The original models have agents standing in line, observing all the decisions ahead of them. This would be the same as having available at the ticket booth a list of all tickets previously purchased. De Vany and Lee improve the model by allowing all agents access to a global source that gives movie shares and each agent can interact only with the agent just ahead in line. While more realistic, this still models a group of agents who never discuss movie options with anyone else before standing in line at the ticket booth.

Another problem I found with the model is that it does not allow for the possibility of an agent ignoring the quality assessment of another agent and deciding to see a bad movie, anyway. To account for this, a small probability of ignoring a bad quality assessment is added to the model. This small probability was added because if an agent decides not to trust another agent's quality assessment, it is probable they will decide that a specific movie may still be good. Further, some people may be so set on seeing a specific movie that they ignore any negative reviews of the movie.

As another aspect that should make the model more realistic, my model allows agents to interact with other agents before going to the movie. Like the previous model, I will be using a multi-agent model in which each agent has access to a personal signal, public information on movie share, and local information gathered from interacting with the agent directly ahead in line. Additionally, agents will have the chance to interact with a random number of other agents. With whom they interact is determined spatially; agents move around a neighborhood and may interact with agents they run into. At every time step, each agent will be allowed to interact with another two agents. The longer it takes an agent to go to the movie, the more agents he will interact with.

This paper investigates how adding spatial movements and agent interaction, and to a small degree adding a chance of ignoring another's quality assessment, affect the formation and choice distribution of an information cascade. These changes can help to explain what leads to certain movies becoming great hits and others going bust, sometimes independent of the movies quality.

Chapter 2

The Approach

2.1 Base Model

The basic model of an information cascade was presented by Bikhchandani et al(1992) [3]. Their model involved agents using Bayesian updating to make a decision based on their own private signal and the information about the actions of all the agents ahead of them. This is the model that involves agent standing in essentially a long line and each agent observes the decisions of each agent ahead of them. They use this information and their own personal signal to decide between two options. De Vany et al. created an agent-based version of this Bikchandani et al. model which I recreated. The model is as described below.

The model involves 2000 agents going to see one of 20 movies. There is a variable, p, which is defined as the probably that an agents personal signal about the quality of a movie will be correct. Of the 20 available movies, half will be movies of high quality and the other half will be of low quality. The agents are lined up sequentially and, in order, choose which movie they would like to see. Figure 2.1 shows the agents at the start of the simulation.

When its an agents turn to see a movie she will have three pieces of information to help

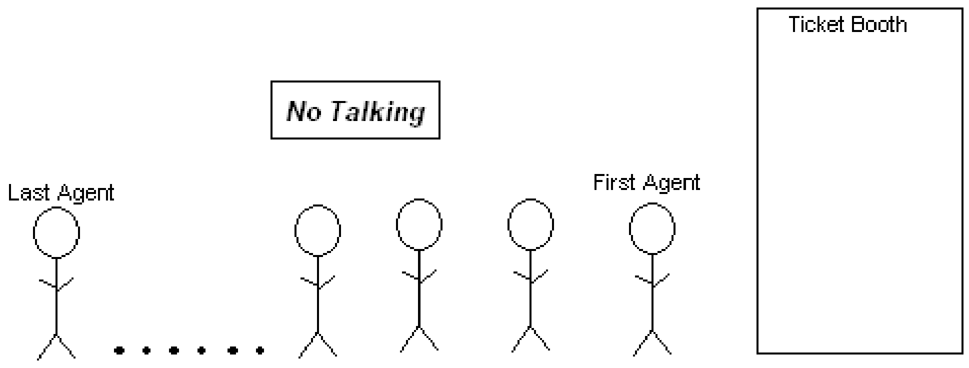

Figure 2.1: Base Model Agent Start Up

her make a decision. The first is a public signal in the form of the market shares of the entire available set of movies. This provides a quantity signal, and to a certain extent a quality signal as well since the movies with the higher market shares are more likely to be good movies. This fact is due to the p value of the agents, since the personal signal is above .5, the agents are more likely to guess the true value of the movie. Of course, how accurate this signal is increases as the p value increases. Also, this gives a hint as to the decisions of all the agents ahead of her. Secondly, the agent is shown the movie viewing decision of the agent directly ahead of her, but of no one else. So the agent doesnt see the decision of everyone in the line but only of the agent ahead of her. The previous agent's decision will either be to see a specific movie or to not see a movie. Finally, the agent will have a personal signal about the quality of the movie the previous agent saw. The probability of the personal signal being correct is set by the variable p and is varied between simulations. When the simulation is set to begin, all the movies are provided one agent who saw the movie. The first agent in the line of agents is then allocated at random to a movie, and then the following logic sets in.

If the previous agent, agent i-1 saw movie m, then agent i will check her own personal signal. If the personal signal is high then the agent will see movie m. If the personal signal is low, then the agent will flip a coin and either choose to not see a movie or pick a movie based on market shares. If the agent chooses to pick a movie based on market shares, the movie will be picked based on a weighted average of the market shares. This means that the movies with higher market shares have a higher probability of being picked. If agent i-1 did not see a movie, then agent i will check her own private signal. If her private signal is low, meaning she doesnt believe the movie is a good movie, then the agent will choose to not see a movie. If her signal is high than she will flip a coin and either not see a movie or pick a movie based on market shares.

2.2 Extended Model

The basic idea behind the extended model is to simulate a neighborhood where agents interact with one another. There are m numbers of movies available which the agents in the neighborhood will discuss with one another throughout the simulation. Each agent will have one opportunity to go the movies, where they will make their final decision about which movie they want to see using the information gathered through interacting with other agents, the decision of the agent directly ahead of them, and their own personal signal.

In my modified model, agents choose between m movies. Agents choose a movie to view sequentially, with one agent choosing for each step of the simulation. There are two special variables for this simulation. P is again defined as the probability that an agents personal signal about the quality of a movie is correct. There is also a value pi, which is defined as the probability that an agent will believe another agents value, which forms the agents opinion, about the quality of a movie. The order for the agents to see a movie is randomly generated

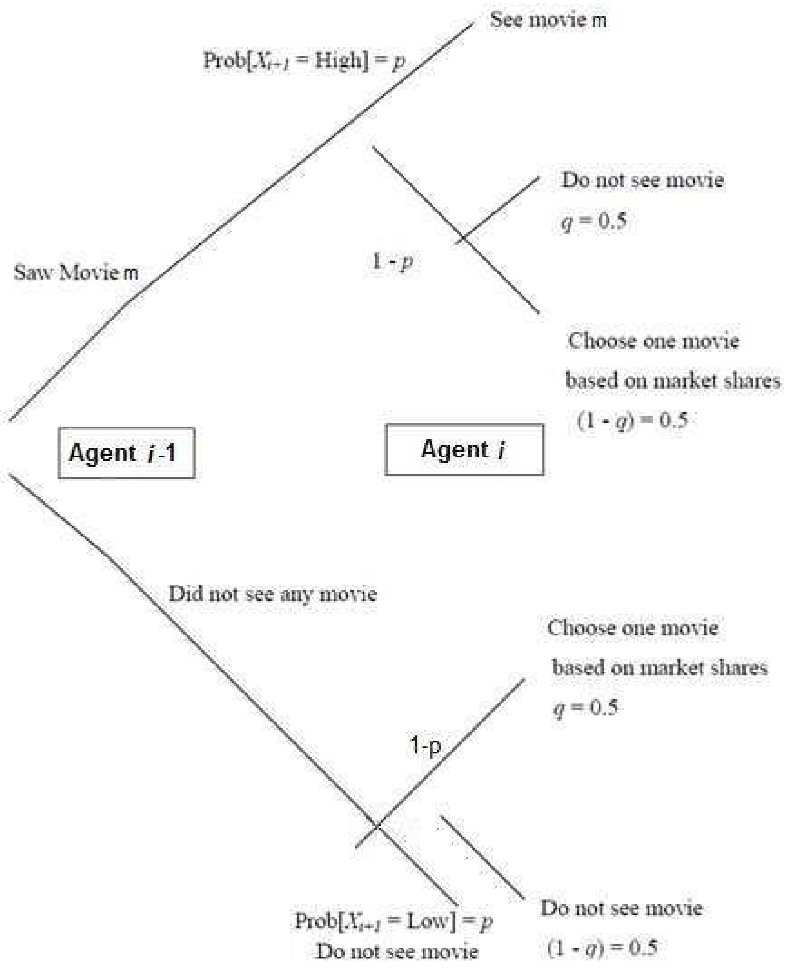

Figure 2.2: Base Model Agent Decision Tree

at the beginning of the simulation. In the beginning of the simulation, agents are placed in houses, more specifically a specified point on the 2D plane. In the neighborhood, the agents are evenly placed among the houses with any extra agent being placed in the first house. The houses are in two vertical rows with a sidewalk running vertically next to the house on the side closest to the other row of houses. While there is no street in-between the two houses, agents cannot communicate with agents on the sidewalk horizontal to them. Figure 2.3 shows an example of the states of the agents. In this example there are 20 starting locations and 40 agents. On the left is the starting location. The squares represent the starting locations of the agents. To the right are the agents after the simulation has begun. They are moving around the neighborhood and multiple agents may be in the same spot at the same time.

The starting place of each agent is determined by the house they are assigned to. This starting location is just a starting point; the agent will continue to move around the map and doesnt have any special connection to this starting point. Movements for these agents go as follows: agents have a 50 percent chance of moving forward one space. They have a 30 percent chance of moving backwards one space. Their final probability of 20 percent is to cross the street, they will move the horizontally. So if the agent is on the left side of the street, she will movie right one space. If she is on the right side of the street, she will move left one space. Agents must stay in bounds, defined as the space of the two rows of houses and sidewalks that belong to the agents specific neighborhood, plus two additional extra spaces on the top and bottom of the neighborhood. Each agent moves one space for each time step.

Each agent keeps an array of their own personal values for all the available movies. While not at a movie, an agent wanders around the neighborhood according to the description above to gather information about the quality of the available movies. For each

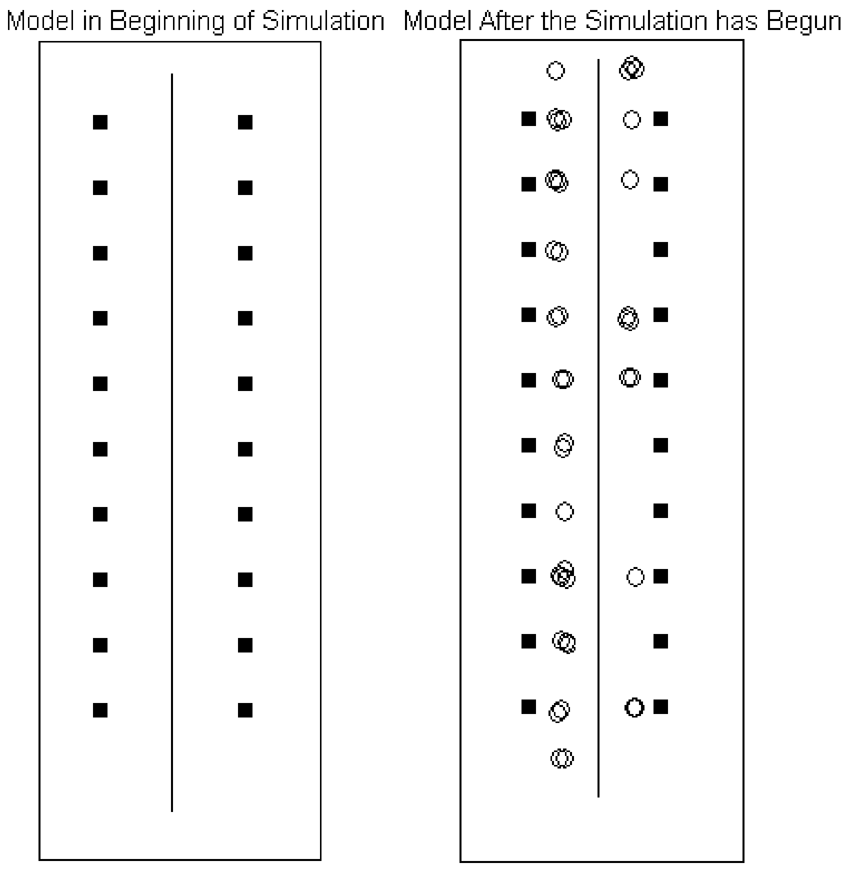
Model in Beginning of Simulation
Model After the Simulation has Begun

Figure 2.3: Extended Model Agent Setup and Movement

simulation, there is one agent per a movie who starts out in the neighborhood who already has seen a movie. This agent never goes to see a movie again but instead moves around the neighborhood passing information on to other agents. As a result every agent who passes on information about a movie got her information from a source that eventually leads to an agent who has seen a movie. This helps spread information at the beginning of the simulation immediately instead of waiting for agents within the simulation to see a movie to begin passing on this information. These agents also simulate critics and other people who see a movie early at a preview.

If it is not the agents turn to see a movie, then the agent seeks to communicate with other agents in the same coordinate spot. Each agent only speaks to two other agents per time-step. An agent communicates only with the agent as it was at the end of the previous time-step. In this case, multiple agents can communicate with a single agent, and these agents will be communicating with the same state of the agent. After this information gathering occurs, the agents then moves and update their information to share with other agents. Then the next time step occurs. The simulation runs until each agent has a chance to see a movie. So given n agents, there will be n steps in the simulation. Each agent starts out with a value of -1 for each movie, meaning the agent has not heard of the movie yet. When an agent meets another agent, they learn about all the movies the other agent has heard about. When an agent first hears about a movie their personal value for the movie is 10, meaning neutral. The agent then has to make a decision of whether or not to increase or decrease their personal value for this movie. These personal movie values are later used to make a decision about what movie the agent will see.

If the other agents personal value for this movie was 10 or greater, then they have a high value of this movie. Otherwise their value is low. If the other agents value is high, then the agent will decide, with probability pi, if she believes the other agents value. If she does then she will choose to increase her personal value of the movie by 2. Otherwise she will check the market shares and see if the movie is in the top 5. If the movie is in the top five but has no views, which is likely to occur at the beginning of the simulation until several different movies have been viewed by agents at the movie, then the agent will check her personal signal. If her personal signal is high, which is correct with probability p, she will decide to increase her personal value of the movie by 1 and if her signal is low she will choose to decrease her personal value for the movie by 1. If the movie is in the top five and does have vies, then the agent will choose to increase her value by 1. If the movie is not in the top 5 movies according to market shares, the agent will choose to decrease her value by 1, regardless of her personal signal.

If the other agents personal value for the movie was low (a value less than 10) then the agent will decide, using value pi, if she believes this other agents value. If she does, then she will decide to decrease her personal value for the movie by 2. Otherwise she will check the market shares to see if its in the top 5. If it is, then she will choose to increase her value by 1. Again, if its in the top five but have zero agents who have seen it, and then the agent will look at her personal signal. If its high then she will choose to increase her personal value by 1, otherwise she will decrease it by 1. A figure of this decision process is shown below.

After an agent makes the decision of whether to increase or decrease her personal opinion of the movie, she does so according to the following method. If the agent decides to increase her personal opinion of a movie and the other agents personal value was high, then the agents personal value will increase by 2. If the agent decides to increase her personal signal but the other agents personal signal was low, the agent will only increase her personal opinion by a value of 1. On the other hand, if the agent decided to decrease her personal opinion and the other agents personal signal was low, the agent will decrease her personal value by

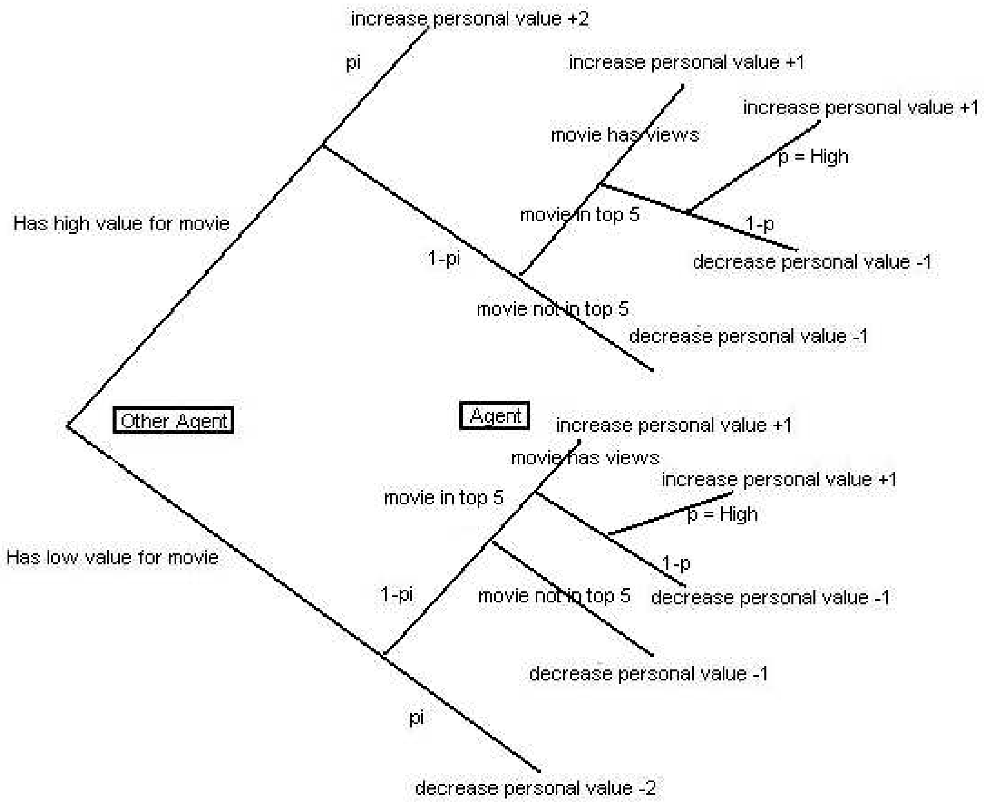
increase personal value +2
pi
increase personal value +1
increase personal value +1
movie has views
p = High
Has high value for movie
movie in top 5
1-p
1-pi
decrease personal value -1
movie not in top 5
decrease personal value -1
Other Agent
Agent
increase personal value +1
movie has views
increase personal value +1
movie in top 5
p = High
Has low value for movie
1-p
1-pi
movie not in top 5
decrease personal value -1
decrease personal value -1
pi
decrease personal value -2

Figure 2.4: Agent Movie Preference Decision Tree

2. If the agent decided to decrease her personal signal and the other agents personal signal was high, then the agent will decrease her personal opinion by 1. The values an agent can assign to a movie range from between 0 and 20 with 0 being the lowest possible value and 20 being the highest.

The way agents communicate to one another means the cascade will occur in the information the agents spread to one another. For example, when deciding whether or not to increase or decrease a movie value and agent may increase their personal value even if their personal signal is low if the other agent says the movie is good and the movie is in the top 5 market shares. In other words, there is a potential cascade for each of the movies in the area of the information agents have about its quality.

At the beginning of a time-step, each agent who has not seen a movie checks if its the agents turn to go to the movies. If so, the agent will remove herself temporarily from the neighborhood, the agent will not be interacting with any other agents. At the movies an agent has two sources of information in which to choose which movie to see. The first is from the media in the form of market shares. This again provides a glimpse of the selections of all the agents ahead of her and in the same regard provides a quality signal. The agent also gets to see the decision of the agent ahead of her. If agent i-1 did not see a movie, then agent i evaluates her own personal signal. This signal has the chance p of correctly conveying the true value of the movie. If the value is low then agent i will choose to not see a movie. Otherwise there is a 50 percent chance agent i will choose to not see a movie or choose the movie with the highest personal value. In the case multiple movies share the highest personal value; one will be chosen at random to be viewed. If agent i-1 saw movie j, then agent i evaluates her own personal signal. If the signal is high then the agent will see movie j, otherwise there is a 50 percent chance agent i will choose to not see a movie or choose to see the movie with the highest personal value. After the agent has selected the movie, the agent will be placed back in the neighborhood, and will communicate with other agents in future time-steps.

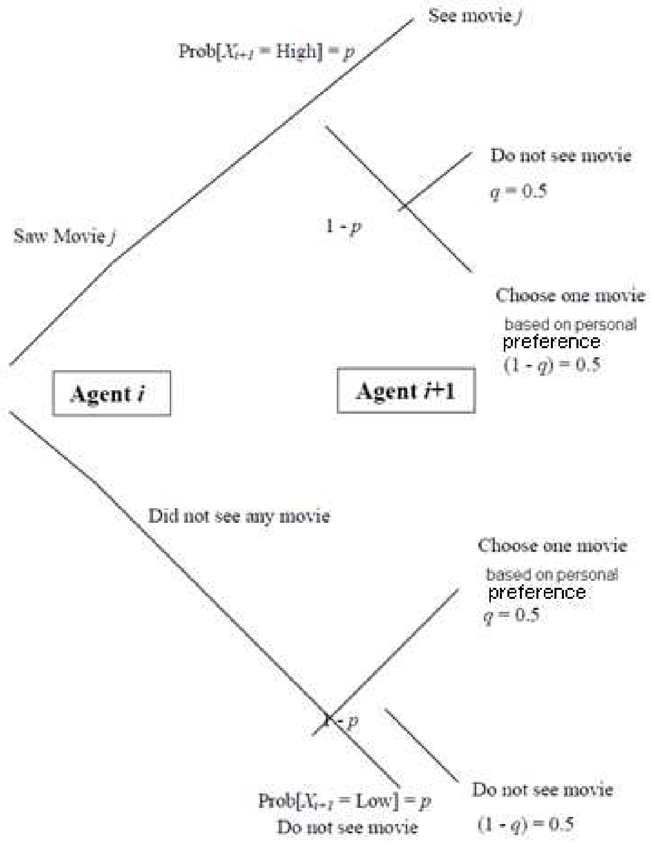

Figure 2.5: New Model Agent Movie Decision Tree

Chapter 3

Base Model Results

3.1 Analysis of Correct Choices

For our model that had 20 movies available for agents to see, we collected two main data points. The first is c, the number of correct choices over the entire sequence. This data helps us determine if higher values of p do increase the likelihood of a correct information cascade. As shown in the table and the graph below, there definitely is a correlation between higher p values and higher mean of correct choices. The other variable of interest is d. This is the mean value of correct choices when you exclude the agents who chose not to see a movie. The variable d is used to help compare this approach to the other approach where agents interact and will be discussed later.

These results make sense because as the agents become better at determining if a particular movie is good, they will move towards cascading on a good movie which leads to a higher chance the agent will see a good movie. If the previous agent saw a good movie, then the current agent has a p chance of seeing the good movie, and a .5(p) chance of not seeing a movie and the same chance to pick one from market shares. So if agents are cascading

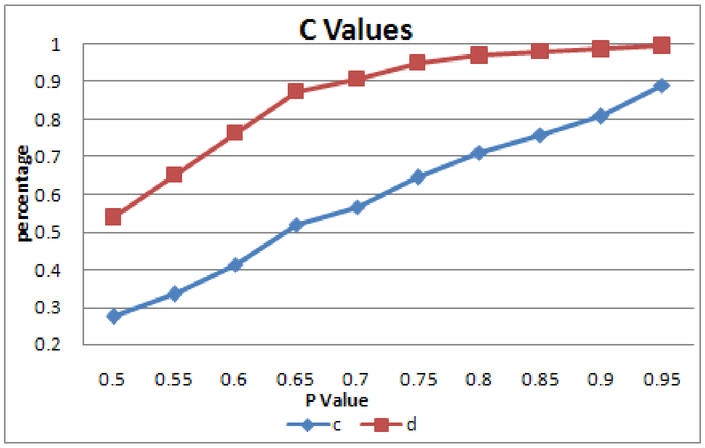

Figure 3.1: Base Method C and D Values

Table 3.1: Correctness of Choice for Base Method

p Value	C	D
0.5	0.274529	0.539771
0.55	0.334659	0.653182
0.6	0.412987	0.764045
0.65	0.518987	0.873395
0.7	0.567641	0.906972
0.75	0.645289	0.950468
0.8	0.710766	0.970843
0.85	0.758819	0.980487
0.9	0.807544	0.987438
0.95	0.890972	0.996431

on a good movie, then the agents simply have a higher chance of see a good movie by just going with the cascade. The values of D clearly show that as p increases agents become better at choosing good movies when they choose to see a movie. Due to both the D and C values shown in the table, my results agree with the first result of the BHW model that as the p value increases the chances of a correct cascade increases.

3.2 Cascade Analysis

The BHW model also predicts that the probability of a cascade is almost 1. In fact, they predict the probability of no cascade drops quickly after the number of agents increases over 5. My results actually agree with the De Vany and Lee model that this is not the case. Below I show the graphs of the movie choices for when p=.55 to show a low accuracy case, and p=.95 for an example of a high accuracy case. In the graphs, the vertical axis corresponds to the 20 potential movie choices. The horizontal axis corresponds to time, which evolves from left to right. Each movie choice is indicated by a box. A value of -1 indicates the agent chose to not see a movie.

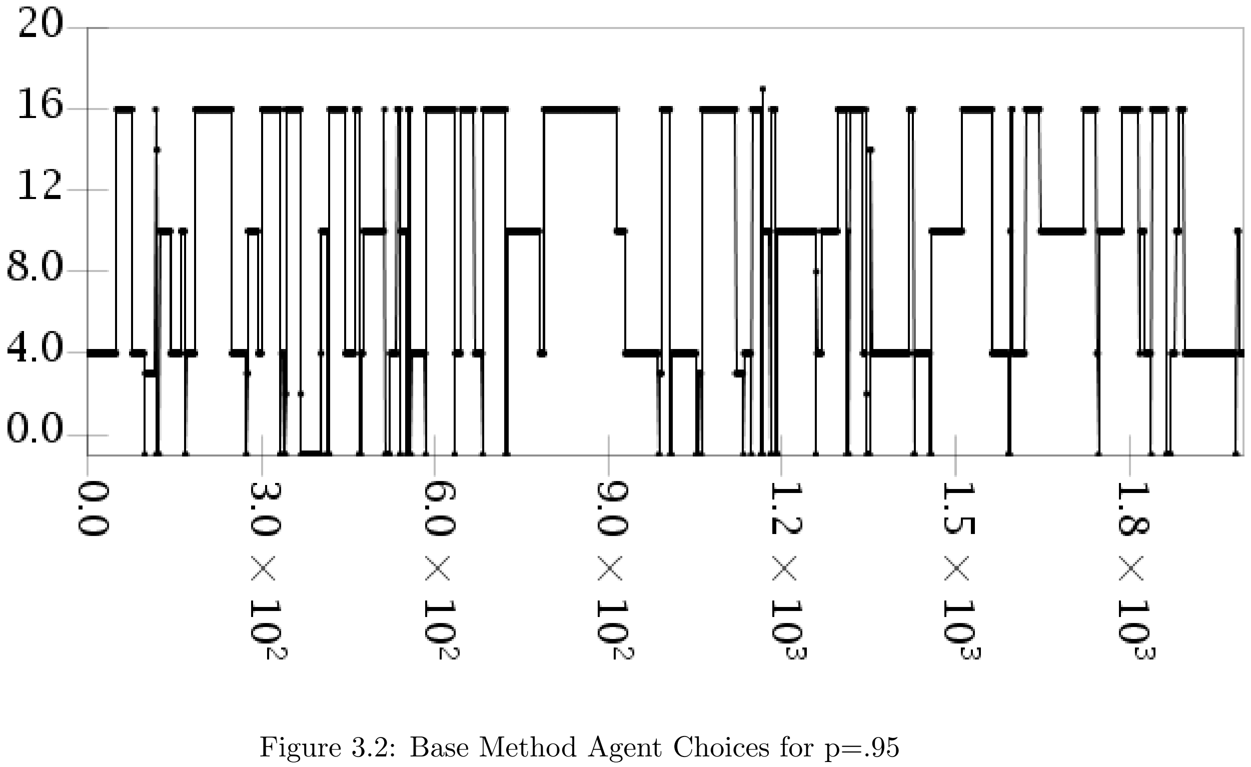

Figure 3.2: Base Method Agent Choices for p=.95

Figure 3.2 shows a high probability case, where p =.95. In this case there is no large

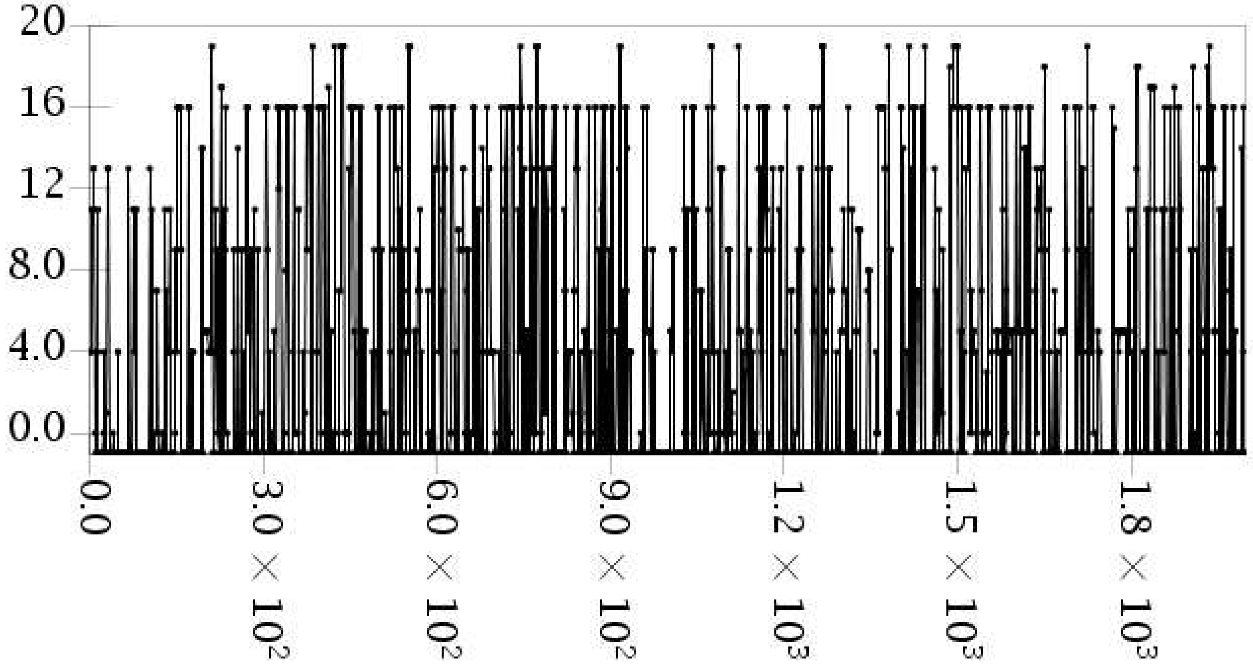

Figure 3.3: Base Method Agent Choices for p=.55

cascade but instead there are several small cascades. Overall the agents seem to be focusing their decisions on 3 different movies, with a jump to any other movie quickly being adjusted back to one of the three more popular ones. The shared focus on the three different movies is probably the result of the decision to force the agent to pick a movie different from the previous agent if this agent chooses to pick a movie from market shares. The results for this high p value found by De Vany and Lee show an overall cascade on a single value with intermediate jumps that would quickly jump back to the single popular value. The rule I added most likely prevents a single movie from getting the vast majority of market shares by allowing other movies to gain market shares by being picked when the top movie is not available to be picked. This figure also shows the ability of a single agent to break a cascade and lead a cascade in another direction. The figure also shows the power of the global information in the form of market shares to pull information back to a previous cascade once market shares are sufficiently uneven.

Figure 3.3 shows the low probability case, where p = .55. As can be seen, choices jump all over the place, which was the case in all the simulations that were run for low p values. Cascades do seem to form; by they are significantly more fragile than for higher p values. This model confirms the finding of the De Vany and Lee model and shows that for high accuracy, information cascades do appear but are fragile. The model differs in that for lower values of p, cascades do occur but are more fragile than for the higher p values.

An important question for movie marketers is, do information cascades occur when there are multiple movies available for a consumer to watch or is the revenue spread out amongst all the different movies? Movie marketers depend on an information cascade effect to increase the revenue for their particular movie. So now we must determine if a cascade has occurred. Different studies have shown that the distribution of motion picture box office revenues is well represented by a Pareto-Levy(stable) distribution with infinite variance.

This distribution is a heavy tailed distribution, meaning it has more probability mass on the extreme outcomes than a normal distribution has. A way to deal with this distribution that was suggested by De Vany and Lee [5] is to examine its upper tail by looking at the top 10 percent and measure the weight in the upper tail. The value of the weight of the tail can be found in the index α.

I estimated the tail index by applying least squares regression to the upper 10 percent of the observation in question. In the upper tail, a Levy stable distribution is asymptotically a Pareto distribution. It takes the following form:

$P[X > x] = x^{\alpha}, forx > k$

Where x is the number of entries and k is a large number. A low value of α corresponds to a slow decay of the tail and is referred to as a heavy tail. This means the information is more clumped together. In regards to movie market shares, it means that the market shares are focused more on a few movies that have a large grab of the market. A higher value of α corresponds to a light tail which means most of the data can be found more spread out. In regards to market shares, this means that the total market shares are more evenly spread out amongst all the movies. To gather this data I ran the simulations with 200 movie choices. I then took the top 20 movies and fit the information to the Pareto distribution and recorded the α value. The results from the base model are shown below in graph form.

This graph show a clear trend of higher p values corresponding to lower α values. This agrees with the findings of De Vany and Lee of higher p values meaing a heavy tail. This confirms the conclusion from the graphs that higher p values produce less fragile cascades. Therefore there is a positive relationship between heavy tailed distributions and information cascades given a high enough value of p.

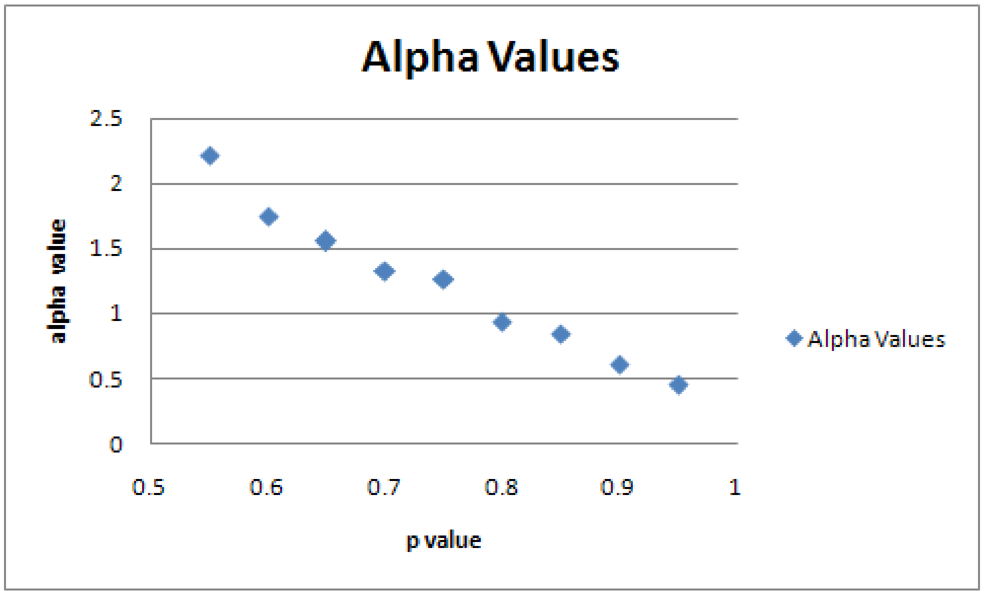

Figure 3.4: Base Method Alpha Values

Chapter 4

Extended Model Results

4.1 Analysis of Correct Choices

We now look at how allowing agents to interact with one another before they make their final movie viewing decision affects the creation of an information cascade. Since this simulation requires two separate variables, a p and pi, we ran 20 separate simulations for each of the possible 10 values of pi and each of the possible values of p. I was looking into how the various values of pi affect the results found in the base results. To look at this effect, I focus on the data for the results from p=.55 and p=.95 to compare to a low and high accuracy case. I will start by looking at the data for p=.55. The chart for this data is shown below. The chart includes the mean C and D value for each value of pi and the difference between the mean values and the mean value from the base method.

I also include a graph showing the c and d values and another graph depicting the difference between the new models average c and d and the new models c and d for all values of pi.

The results show that adding in the pi factor helped agents make a correct decision

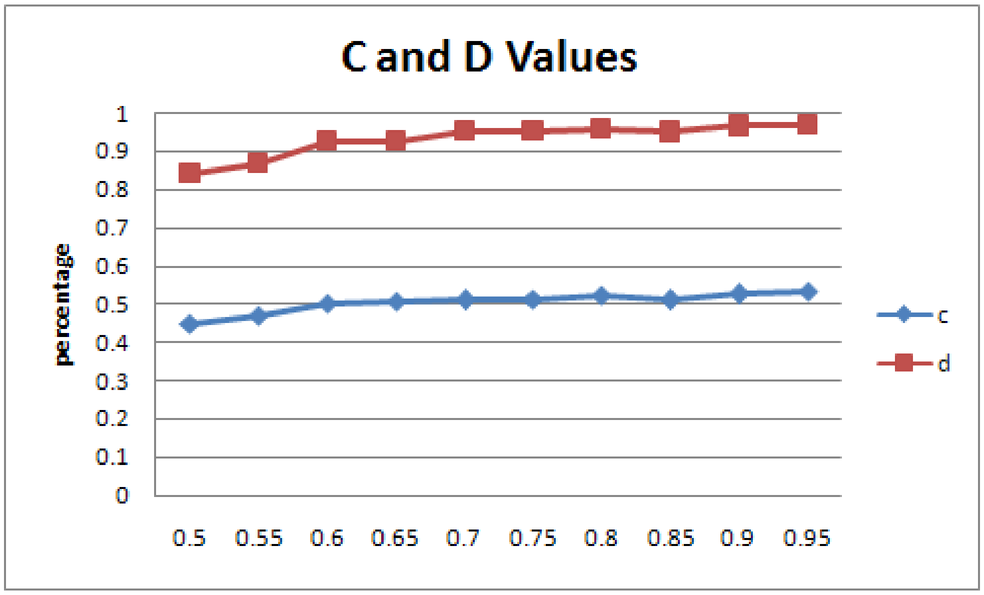

Figure 4.1: Extended Method C and D Value for p=.55

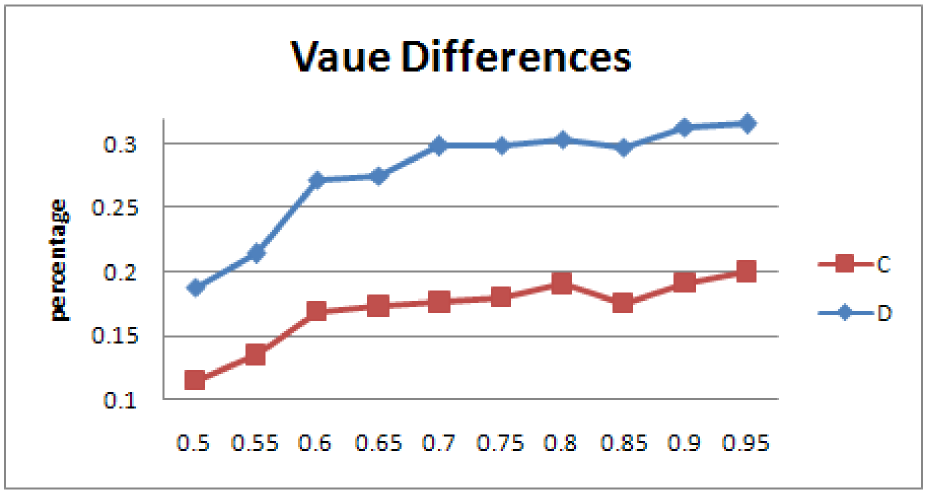

Figure 4.2: Extended Method Change in C and D Value for p=.55

Table 4.1: Correctness of Choice for Extended Model p=0.55

pi Value	C	D	C Difference	D Difference
0.5	0.449956	0.841474	0.115297	0.188292
0.55	0.469187	0.867437	0.134528	0.214255
0.6	0.503665	0.924868	0.169006	0.271686
0.65	0.507238	0.927869	0.172579	0.274687
0.7	0.510464	0.95149	0.175805	0.298308
0.75	0.514462	0.952332	0.179803	0.29915
0.8	0.524888	0.957076	0.190229	0.303894
0.85	0.510338	0.950779	0.175679	0.297597
0.9	0.525688	0.966522	0.191029	0.31334
0.95	0.534689	0.968965	0.20003	0.315783

for all values of pi. Part of the reason for this can be found in the even higher boost the pi factor added to the D variable. When agents needed to pick a movie after deciding to see another movie, they now have more reliable information on which to make a decision. Since the beginning source of the information about a movie is from an agent that actually saw the movie, the information starts out correct. Only through agent's distrust, which occurs more with lower pi values, does this information lose its correct value. The reason for the higher C and D values for the extended model can be found in the beginning of the simulation. For the base mode, in the begining market shares are not well spread out so when an agent has to pick a movie with only the marketshares to help, they have to essentialy guess a movie to see because they lack information. In the new model this is only the case for an agent who has not talked to another information bearing agent. Its possible that in the new method agents talk to one another so agents have useful information sooner in the simulation than in the original model. As a result they start to stick to good movies sooner than in the original method.

Next we look at a high accuracy case, the case when p =.95. The table and graphs are shown below.

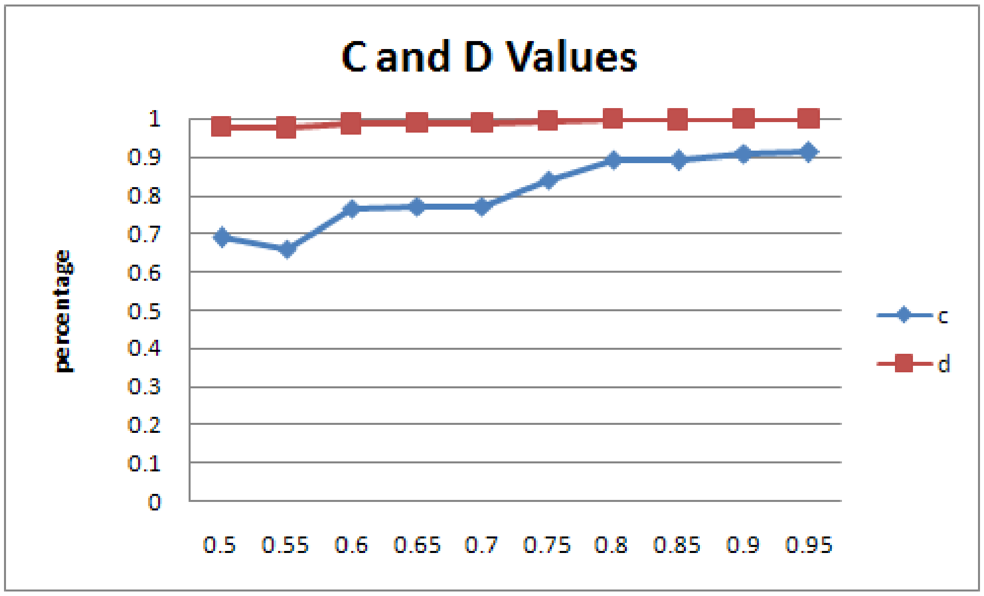

Figure 4.3: Extended Method C and D Value for p=.95

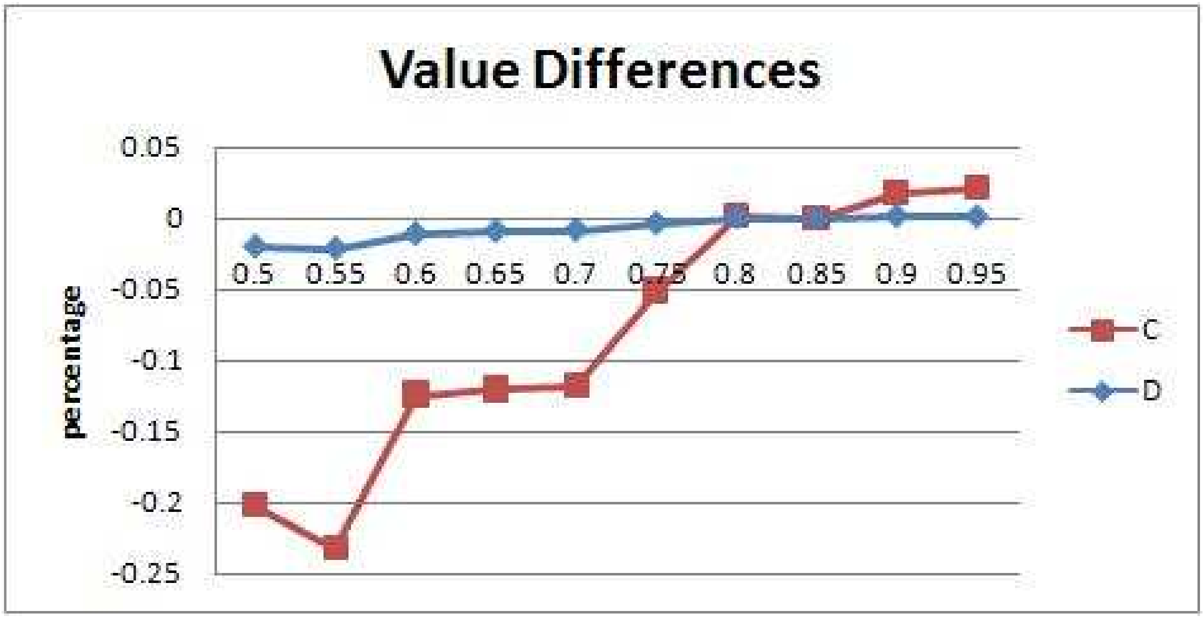

Figure 4.4: Extended Method Change in C and D Value for p=.95

Table 4.2: Correctness of Choice for Extended Model p=0.95

pi Value	C	D	C Difference	D Difference
0.5	0.688368	0.976545	-0.202604	-0.019886
0.55	0.658895	0.975046	-0.232077	-0.021385
0.6	0.766373	0.985563	-0.124599	-0.010868
0.65	0.77009	0.987432	-0.120882	-0.008999
0.7	0.772198	0.988147	-0.118774	-0.008284
0.75	0.839246	0.992689	-0.051726	-0.003742
0.8	0.892321	0.99704	0.001349	0.000609
0.85	0.890823	0.996458	-0.000149	2.7E-05
0.9	0.907699	0.997935	0.016727	0.001504
0.95	0.911449	0.997574	0.020477	0.001143

In this case, adding in the pi aspect seemed to have hurt the C and D values for all values of pi that are less than p-0.15. This result makes a lot of sense. In the original method, when an agent needed more information, she turned to the market shares which were accumulated by the decisions of other agents who have a 95 percent chance of knowing if a movie is good or not. In the new model, while the agents that start with the movie information are 100 percent correct, as they pass on the information, the information has a less than 95 percent chance of being correctly passed on to the next agent. As a result, the extra information given to an agent when she needs extra information to choose a movie may not be as reliable as the information available to agents in the original model. This fact is confirmed by the D values being lower for all values when pi ¡ p-1.5. Inspecting all results from other p values confirms that the C and D statistic improve for all values of pi >= p-0.1 or p >= p-0.15.

For all values below this threshold, the reason the C and D statistic would be hurt has to do with the quality of information an agent gets when they choose to change the movie to see. With high p values in the base simulation, over time the movie statistics act like a quality signal to the agents because agents are very good at choosing a good movie. Over time this ability to choose a good movie will result in only a good movie getting high movie shares. When pi is a lot lower than p, their personal knowledge of which movie is good is poor, since the information has potentially been passed through several agents who have low trust in other agents. As a result misinformation is spread around and this incorrect information is then used to make a decision that is not as well informed as possible.

4.2 Cascade Analysis

Does the ability of agents to move around and communicate with other agents help or hurt the formation of a cascade? Would helping the agent pick a better movie help the formation of a cascade and if the pi value is a lot lower than the p value, will this hurt the formation of a cascade? To see if a high value of p helps increase the probability of a cascade, I will compare the sequencial graph of agent movie choices for various values. First I will compair two graphs for a low value of p, when p=.55 for pi values of .55 and .95.

These results suggest that for a given value p, increasing the pi value does not increase the likelihood of a cascade; although an intersting trend is shown by the graphs. While the data points jump around, they tend to be constrained on bands more so than the base method. This is probably the result of good information being spread around the agents. The agents share information so each agent has a fairly good idea of which movies they should see and which ones to avoid. The only section of Figure 4.5 that seems more random than the rest and is not on a more defined band of choices is the very beginning. This observation can be explained by the fact that many agents who are chosen to see a movie in the beginning may not have spoken to any agents who have any information about a movie. As a result, they have to make their decision as blindly as agents in the original model have to at the beginning of the simulation until movie shares are sufficiently spread out.

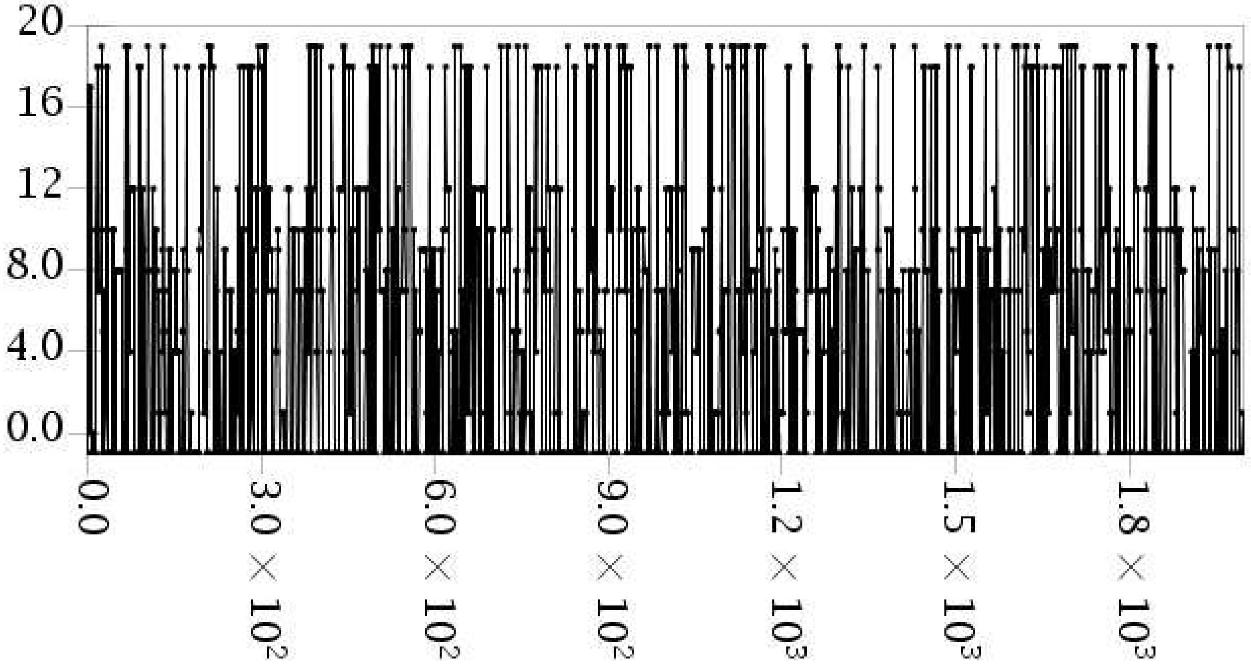

Figure 4.5: Extended Method Agents Choices for pi=.55

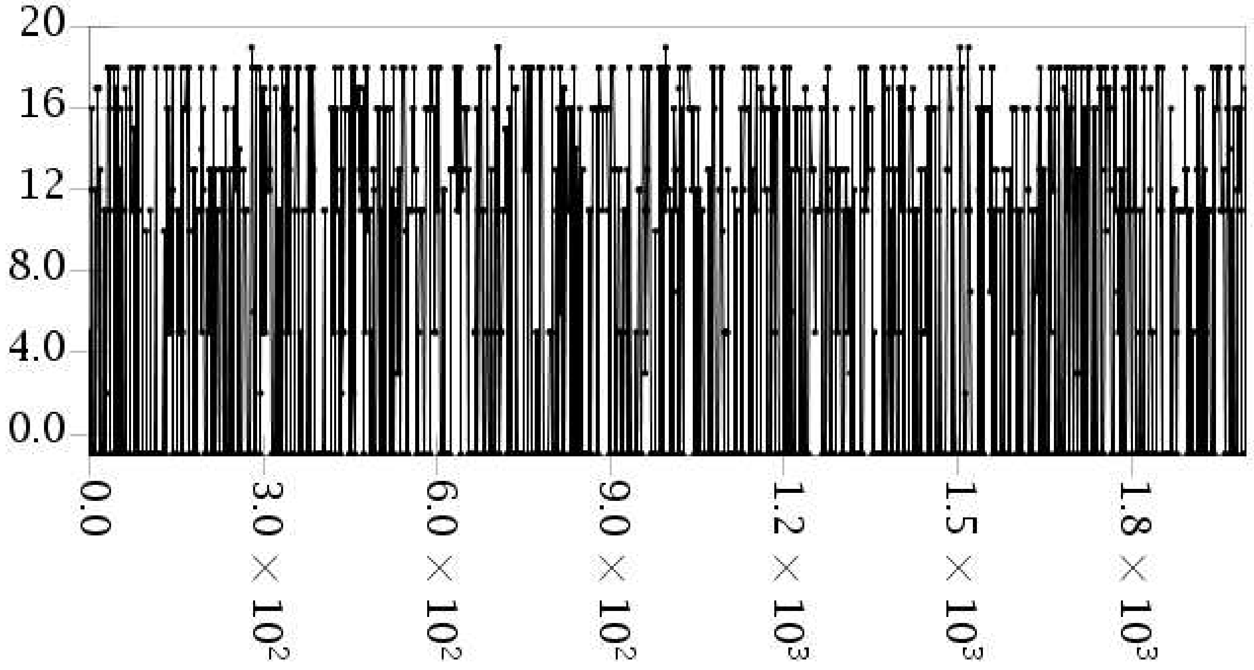

Figure 4.6: Extended Method Agents Choices for pi=.95

Now I will look into a case with high p, when p = .95. Figure 4.5 and Figure 4.6 compare pi values of .55 and .95 respectively.

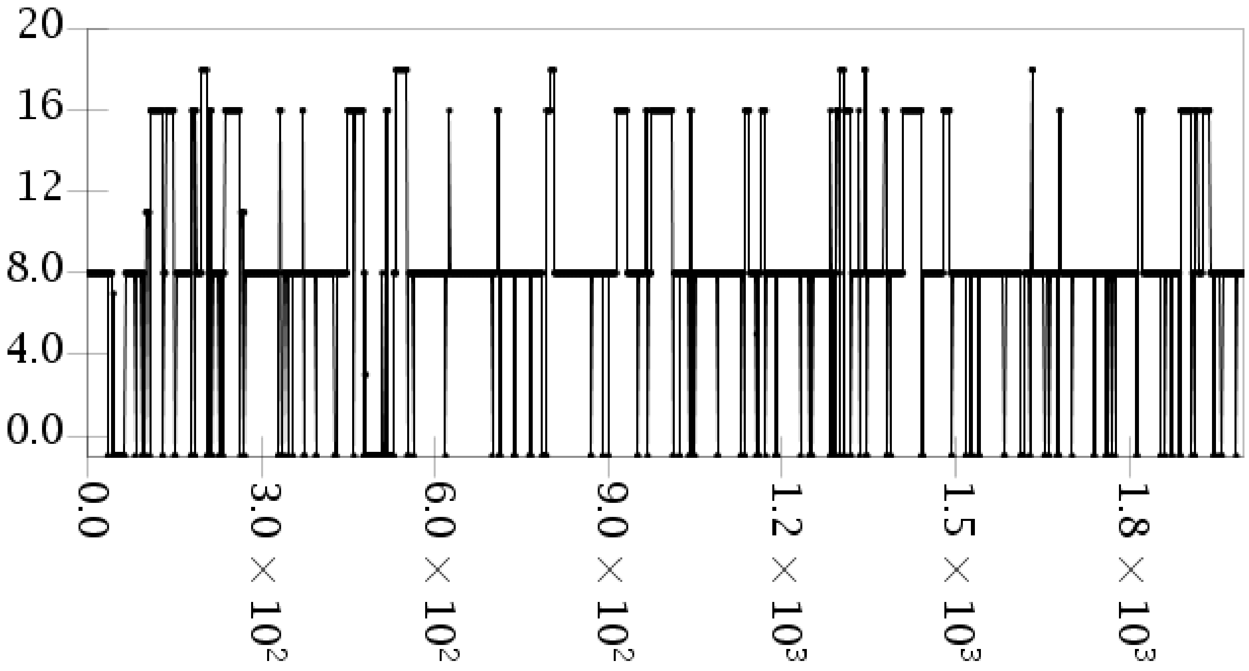

Figure 4.7: Extended Method Agents Choices for pi=.55

Figure 4.7 gives an interesting result. This graph shows several small cascades that tend to focus on two movie choices. The long gaps seen between two movies choices is the result of the movie that started the gap being a bad movie. Since the p value is at .95 most agents correctly guess that the movie is bad which, for the first agent, results in a .5 chance of not seeing a movie. For the rest of the agents their chance of not seeing a movie is .975. What probably caused the choices to mostly stay between two different movies has to do with the initial choices made when the agents were interacting in the neighborhood. As the various agents who have already seen the movie talked to other agents, these agents decided to not believe this agent and incorrectly changed their personal information. These agents

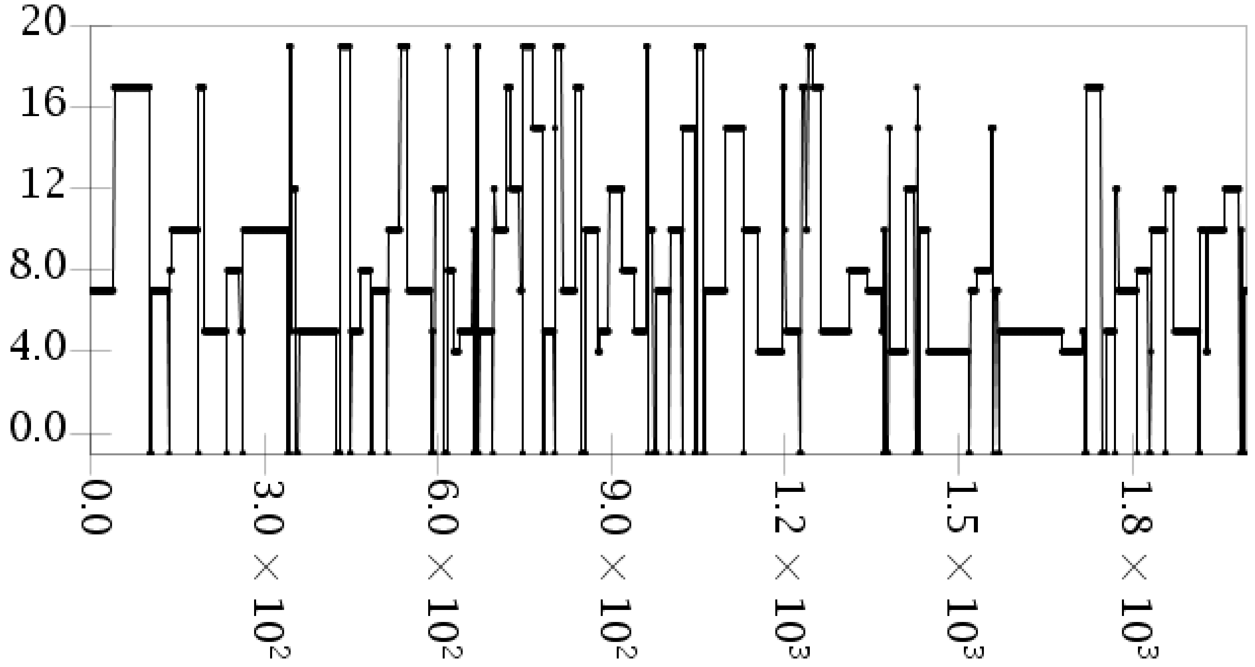

Figure 4.8: Extended Method Agents Choices for pi=.95

then spread this misinformation to other agents. In other words, its possible that a slightly higher number than expected of agents mistook a good movie for bad at the beginning and then spread this misinformation around. It's possible that for most agents, the correct information received in the area of good movies was for the two good movies that make up the two cascades.

Figure 4.8 shows some very small and very fragile cascades occurring, of movies 5,7,10 the lack of any large gaps between any two movies, and the high D value, show that agents are switching between movies of high value. The reason this graph looks more like it is jumping all over the place is because in this case, agents figures out rather quickly in the simulation which movies are good, so when the decision to change the movie to be seen comes up, agents chose from one of almost any of the actually good movies available.

By looking at these graphs, it seems that increasing the pi value alone does have an effect on the formation of cascades. The effect is found in how many movies agents pick between when they must choose a movie. In the low pi value, the probability of having the true knowledge of which movie is good is more uncertain. In the specific case I showed agents picked mostly between two different movies. The exact number changes with different run, but the important fact is that the exact number is more uncertain than for high pi values. In the case of high pi values, agents know with pretty good accuracy and which movies are good. As a result they can choose between this greater numbers of movies.

Now we look at the decay of the tail for our new model. First we look at the α values for when the p value is .55 across all pi values. The values are graphed in Figure 4.9.

We now look into how our new model affects the formation of cascades. To do this we look at the decay of the tail for our new model. First we look at the α values for when the p value is .55 across all pi values. The values are graphed below.

In general as the pi value increased the alpha value increased. This means that as the

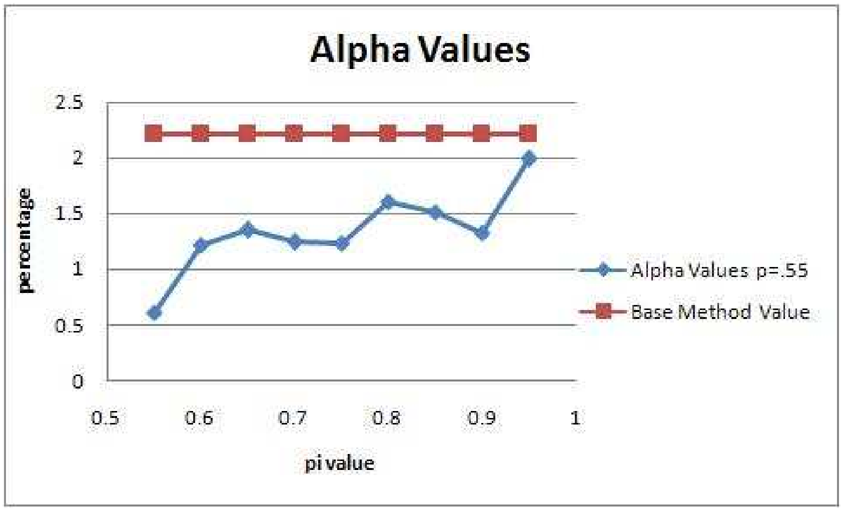

Figure 4.9: Extended Method Alpha Values for p =.55

pi value increase, a fewer number of movies collect a larger majority of the movie viewing. As a result, more movies in the top 20 received only a few views and thus have only a small share of the market since the views are focused more in the top movies. This result indicates that the higher pi values allowed the agents to quickly find the true value of a movie so they picked to see this movie as they switched their movie selection. This result makes sense given the previous data. The basic model showed that higher p values gave lower alpha values, meaning that lower p values mean higher alpha values. Given this context, it makes sense that higher pi values will give higher alpha values due to how the alpha value is found. Since only the top 20 of the 200 movies are used, the higher pi values will have the higher movie shares along more of the top 20 movies. In the case of the lower pi value, agents get conflicting information so the movies agents pick overall are more spread out to include movies that are not good, but mistakenly are thought to be good.

Despite the affects of a higher pi value, in this case the α values always stayed below the

α values from the base model. So overall adding the ability of agent to transmit information to one another appears to help build a less fragile cascade. While no cascades seemed to appear in the images above, this heavier tailed choice may have manifested itself in an extra agent choosing the same movie. So, for example, there may have been a string of three agents choosing the same movie instead of two as shown by in figures 4.10 and 4.11. The figures show 100 movie choices of agents in two different simulations. The graph starts with the choice of the 500th agent to see a movie and records the following 100 choices. Please note that these two graphs do not show the choice to not see a movie sense the graphs are used to loot at extra cascades in the movies being seen.

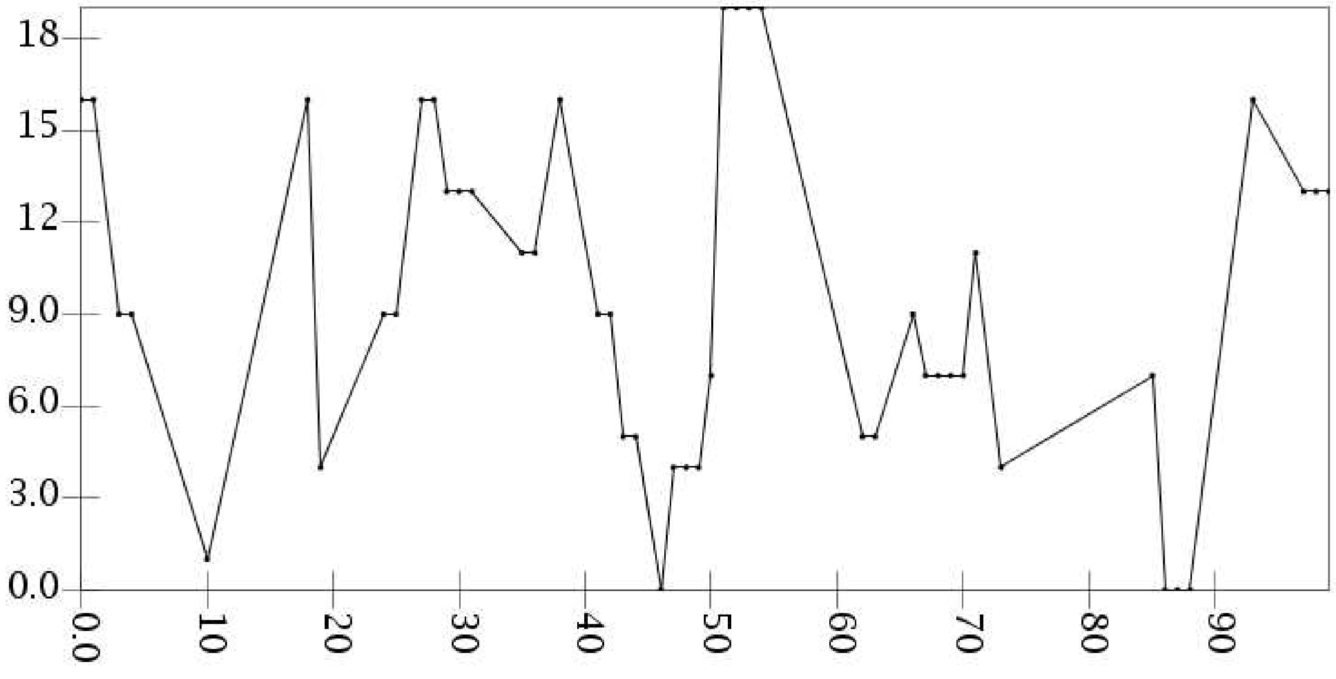

Figure 4.10: 100 Movie Choices from the Base Model with p=.55

This result can be explained by agents making better choices and the fact that if an

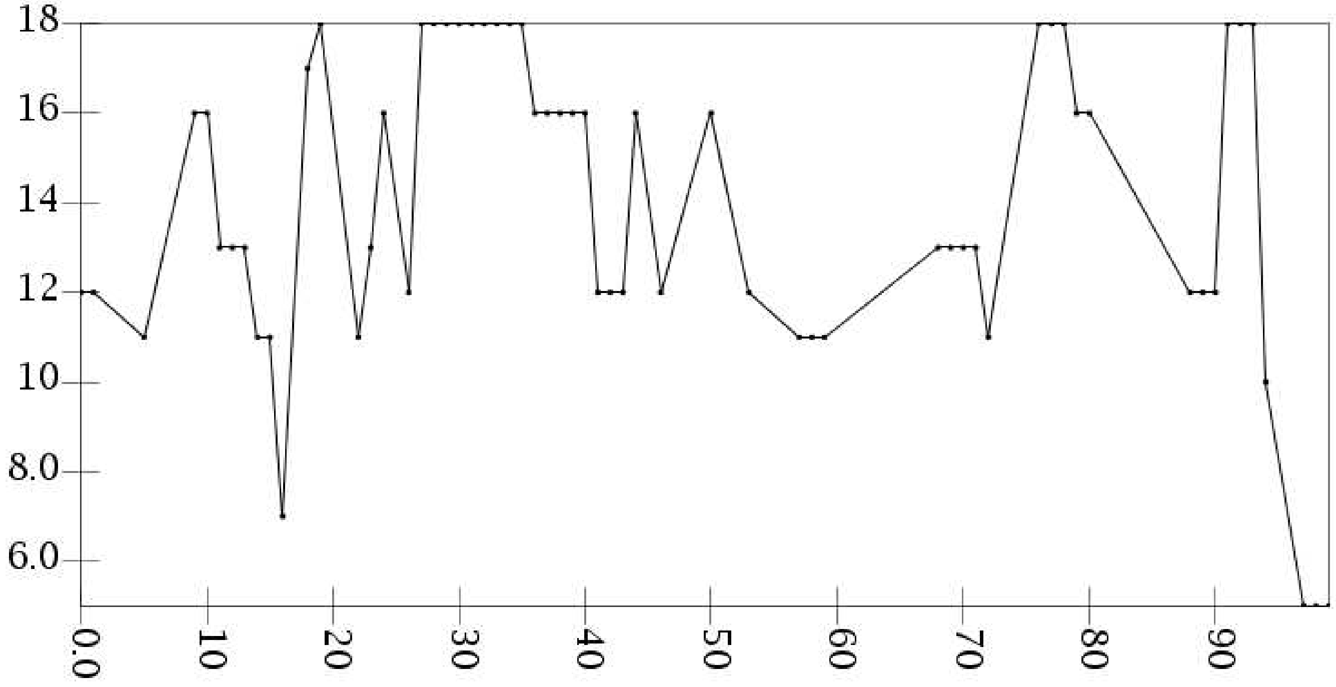

Figure 4.11: 100 Movie Choices from the New Model with p=.55 and pi=.95

agent picked a good movie, there was a higher chance that more agents would follow than if an agent picked a bad movie. While this is probably not technically a cascade, it does produce a herding effect that helps produce a cascade like behavior.

Next we look into the α values for when p=.95.

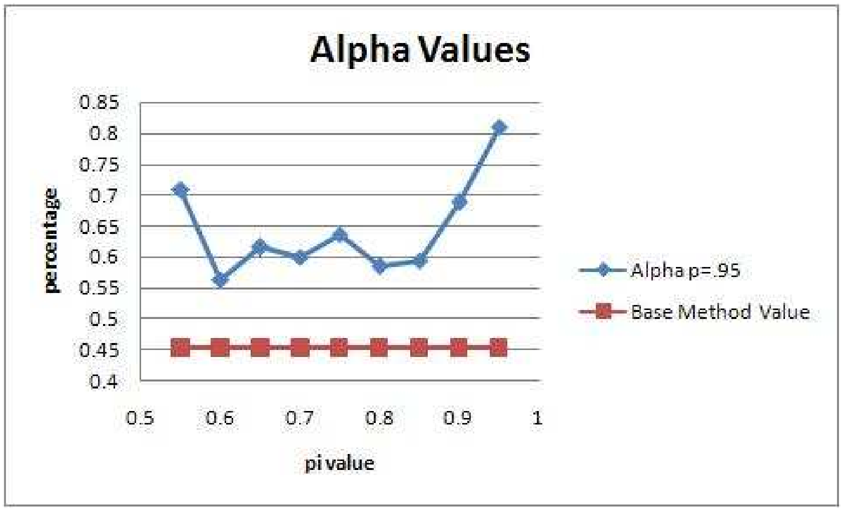

Figure 4.12: Extended Method Alpha Values for p=.95

In this case, the there is no clear result that increasing the pi value for a high p value brings. At the extreme end the α value increased to over .8. In the base model, the alpha value for this p value is 0.45. This means that for higher values of p, allowing agents to speak to one another produces a lighter tailed distribution while for lower values of p, allowing agents to speak to one another creates a heavier tailed distribution. What is interesting is that the highest and the lowest pi values for p = .95 had the highest alpha values, for the middle pi value the α values remained somewhat even. This means that for the lowest and the highest values, cascades became more fragile than for the middle values of pi. All of these new alpha values are higher than the results from the base model, which goes against the previous results. A possible reason for this is that, in the base model, agents would stick more easily to a small collection of good movie, which would quickly dominate the market shares and continue a cascade on that movie. In the new model, agents learn about many different movies and have several movies with high values. As a result, when given the chance to change movies, these agents will pick from a larger collection of movie than from the base model. This would produce higher $alpha values.

The interesting effect of adding the agent communication was its effect on α. In all cases, given all else equal, increasing the pi value will increase the α value, meaning cascades will become more fragile. But when compared to the base model value, adding agent communication can either make it worse or better. For low p values the α value became better, but it became worse for higher p values. This effect is probably the result of when agents pick another movie to see. In the case of the low p, cascades were probably more fragile in the base model because agents had less idea which movies were good and thus when picking another movie the choices would appear more random. In the case of a higher p value, in the base model agents know which movie is good and this movie is probably one of two movies based on market share, so the choice rotates back and forth between the two movies. In the case of agent communication, when agents pick a new movie they pick from any of the good movie, thus creating more diversity in where movie choices jump.

Chapter 5

Conclusion and Recommendations

5.1 Conclusion

Allowing agents to speak to one another before making a final movie viewing decision gives agents another potentially useful piece of information to help them make correct decisions. By taking the average percentage of correct choices for 20 simulations from the base simulation and comparing it to the average percentage of correct choices for 20 simulations for each value of pi given each p value, I found that as pi increased, the percentage of correct choices increased. I found that as a general rule, if the pi value is less than the p-0.1, the agents were better off and made more correct choices. We know this effect is the result of the information gained by talking to other agents by looking at the d value. Since this value excludes the agents who did not see a movie, we know that agents simply chose more good movies with the new method than the old method, given a high enough pi value relative to the p value.

In terms of agent communication on the α value, agent communication can help or hurt depending on how well the base method did in helping agents make correct decisions. This means that agent communication helps more with lower p values than higher values. In the case of lower p values, agent communication helps agents know which movie is good, so they tend to choose between the few movies they know are good instead of guessing on a movie by looking at the market shares which were formed by other agents guessing. In the case of the higher p value, instead of agents focusing intensely on a single good movie, agents focus on one of any of the good movies they know about. As a result the α decreased because agents are choosing between a majority of the good movies.

Does adding agent communication help or hurt the formation of an information cascade? Adding agent communication changes the way a cascade presents itself. The definition of an information cascade states that an agent may make a rational decision that goes against their own personal information. The way the new simulation is set up, the cascades occur in the information the agents spread to one another. For example, when deciding whether or not to increase or decrease a movie value and agent may increase their personal value even if their personal signal is low if the other agent says the movie is good and the movie is in the top 5 market shares. This cascade of information does not produce a clear straight line type cascade over the movie choices of the agents but instead produces bands over the various movie choices of the agents. In a sense, there is a potential cascade going on for each movie in the area the information agents have about the quality of the movie.

5.2 Reconmendations

There are many ways in which this model can be extended for further research. First, as mentioned above, the environment the agents are in can be altered to include neighborhoods and work environments. This change allows the study of how information is diffused. Another factor that can be altered in this sense is the density of the agents. Our simulation had 50 agents per a house initially. You can adjust this so the density is less to model a more typical neighborhood.

Another area where changes could be studied is to alter the preferences of movies. In our simulation we assigned half the movies to be good and the other half to be bad. We assumed that all agents have the same taste in movies and would thus enjoy any of the good movies. In reality not everyone enjoys action movies and some love horror while others will refuse to see it. This preference can be changed by giving agents a movie preference. For example, there could be three types of movies, one which the agent loves, another they dislike, and a neutral one. If an agent sees a movie they dislike, then they will transmit the movie as bad to other agents even if it was a good movie. This change can help investigate if a cascade for movies can occur in an industry where consumers carry different preferences for the offered product.

Another way to make the agents more complex is to vary the pi value for different people they interact with. Most people in the real world place different values on different peoples opinions. In the context of the movie industry, an agent may trust the opinion of a close friend who shares the same movie preference as herself more than a random stranger she meets on the streets. This change will affect how information is transmitted to different agents in the simulation and the accuracy of the transmitted information.

Another area of study is how the diffusion of information affects the various aspects of an information cascade. In this simulation 2000 agents are initially placed in 40 houses so there are 50 agents per house. The decision to make the agents this dense was to ensure information got passed around the agents quickly. There are many other formats that can be investigated. Further experiments could explore different types of environments. For example, agents could be placed in multiple neighborhoods that are isolated. This environment could be extended to allow a common meeting place between neighborhoods, such as a work space where agents from different neighborhoods can interact with one another. Also, in my simulation, agents can only speak to two different agents in a time step. I would imagine that changing this value would have some, even if its only a little, effect on the results.

As we extend the model to simulate more real life experiences, we can get a better idea if information cascades do occur.

Bibliography

[1] Jonathan E. Alevy, Michael S. Haigh, and John A. List. Information cascades with financial market professionals: An experimental study. (18976), 2003.

[2] Lisa R Anderson. Payoff effects in information cascade experiments. *Economic Inquiry*, 39(4):609–15, October 2001.

[3] Sushil Bikhchandani, David Hirshleifer, and Ivo Welch. A theory of fads, fashion, custom, and cultural change as informational cascades. *Journal of Political Economy*, 100(5):992, 1992.

[4] Celen Bogachan and Shachar Kariv. Distinguishing informational cascades from herd behavior in the laboratory. *American Economic Review*, 94(3):484–498, June 2004.

[5] A. De Vany and C. Lee. Information cascades in multi-agent models. (99-00-05), 1999.

[6] Jacob K. Goeree, Thomas R. Palfrey, Brian X. Rogers, and Richard D. Mc Kelvey. Self-correcting information cascades. *Review of Economic Studies*, 74(3):733–762, 07 2007.

[7] Steffen Huck and Jorg Oechssler. Informational cascades in the laboratory: Do they occur for the right reasons? *Journal of Economic Psychology*, 21(6):661–671, December 2000.

[8] Dorothea Kubler and Georg Weizsacker. Limited depth of reasoning and failure of cascade formation in the laboratory. *Review of Economic Studies*, 71(2):425–441, 04 2004.

[9] C. Oberhammer and A. Stiehler. Does cascade behavior in information cascades reflect bayesian updating? (2001-32).

[10] Ivo Welch. Herding among security analysts. *Journal of Financial Economics*, 58(3):369–396, December 2000.

Appendix A

Base Model Agent

```
package edu.trinity.cs.mas.informationCascade;
import java.awt.geom.Point2D;
import java.awt.geom.Rectangle2D;
import java.util.ArrayList;
import java.util.List;
import Communication.KQML;
import edu.trinity.cs.mas.Entity;
import edu.trinity.cs.mas.EntityFinder;
import edu.trinity.cs.mas.Renderer;
import edu.trinity.cs.mas.frontend.Log;
import edu.trinity.cs.mas.services.EntityData;

public class SimpleAgent implements Entity{
        public SimpleAgent(int apos,int aID, int Type, double Prob, int[] agentlist){
                xloc = 10;
                yloc = 10;
                agentID = aID;
                type = Type;
                prob = Prob;
                arraypos = apos;
                findMediaAgent = true;
                AgentList = new int[agentlist.length];
                AgentList = agentlist;
                findMediaAgent = true;
                counter = 0;
        }
        public void gatherData(List<Entity> list, EntityFinder finder) {
                        for(int i = 0; i < list.size(); i++){
                                if(list.get(i) instanceof MediaAgent){
                                                int [] movieValue = ((MediaAgent)list.get(i)).ge
```

```
                                        int movieNum = ((MediaAgent)list.get(i)).getMovie
                                        double[] marketShares = ((MediaAgent)list.get(i))
                                        int movieAssign = ((MediaAgent)list.get(i)).getla
                                        int recentMovieSeen = ((MediaAgent)list.get(i)).g
                                        decision = new Decision(movieValue, movieNum, mar
                                        movieToSee = decision.makeDecision(prob, movieAss
                                        ((MediaAgent)list.get(i)).seeMovie(movieToSee, mo
                                counter++;
                                return;
                        }
                }
                System.err.println("Media Agetn Not Found turn = " + turn + " agent ID =
        counter++;
}
public void reset(){
        findMediaAgent = true;
        counter = 0;
}
public Rectangle2D getBoundingSpace() {
        return new Rectangle2D.Double(-20,-20, 20, 20);
}
public Log getLogData(int i) {return null;}
public List<Entity> getNewlyCreatedEntities() {return new ArrayList<Entity>();}
public Point2D getPosition() {return new  Point2D.Double(xloc,yloc);}
public Renderer getRenderer() {return null;}
public int getSize() {return 0;}
public double searchRadius() {return 50;}
public void setArrayPosition(int pos) {arraypos = pos;}
public void setID(int x) {agentID = x; }
public void setLocation(int x, int y) {
        xloc = x;
        yloc = y;
}
public int getID() {return agentID;}
public void update(int timestep) {      timeStep++;}
public List<KQML> getMessages() {return new ArrayList<KQML>();}
public void packageData(EntityData ed) {}
public void recieveMessage(KQML message) {}
public void recieveMessages(List<KQML> messages) {}

private int arraypos;
private int xloc;
private int yloc;
private int agentID;
private Decision decision;
```

```
        private int type; //type = 1 means quality decision making, 0 means decision made withou
        private double prob;
        private int movieToSee = 0; //this the movie the agent will see
        private int timeStep = 0;
        private int[] AgentList;
        private boolean findMediaAgent;
        private int turn;
        private int counter = 0;

        private static final long serialVersionUID = 2378493020539472835L;
}
```

Appendix B

New Model Agent

```
package edu.trinity.cs.mas.informationCascade;
import java.awt.geom.Point2D;
import java.awt.geom.Rectangle2D;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import Communication.KQML;
import edu.trinity.cs.mas.Entity;
import edu.trinity.cs.mas.EntityFinder;
import edu.trinity.cs.mas.Renderer;
import edu.trinity.cs.mas.frontend.Log;
import edu.trinity.cs.mas.services.EntityData;

public class Simple2DAgent implements Entity{
        public Simple2DAgent(int x, int y, int apos,int aID, int Type,int bNum, double Prob, dou
                                int movieAssig, int mNum, int uBounds, int lBounds, int lXbounds
                                boolean work, int[] agentlist, int movieseen2){
                xloc = x;
                yloc = y;
                initialX = x;
                initialY = y;
                agentID = aID;
                type = Type;
                prob = Prob;
                arraypos = apos;
                yourTurn = false;
                buildingNum = bNum;
                if(buildingNum%2 == 0 )lSideOfStreet = true ;
                else lSideOfStreet = false;
                moviesKnown = new int[mNum];
                agentMovieValues = new int[mNum];
                movieseen = movieseen2;
                agentNum = agentlist.length;
                pi = PI;
                home = new Point2D.Double(x,y);
                lowerBounds = lBounds;
                upperBounds = uBounds;
                tempagentMovieValues = new int[agentMovieValues.length];
                lowerXbounds = lXbounds;
                upperXbounds = uXbounds;
                goToWork = work;
                tempMoviesKnown = moviesKnown;
                reset();
        }
        public void reset(){
                xloc = initialX;
                yloc = initialY;
                timeStep = 0;
                for(int i = 0; i < agentMovieValues.length; i++){
                        agentMovieValues[i] = 0;
                        moviesKnown[i] = -1;
                }
                if(movieseen != -1){
                        moviesKnown[movieseen] = 10;
                        movieSaw = movieseen;
                }
                counter = 0;
                findMediaAgent = true;
                if(agentID == -1){
                        turn = -1;
                }
                int m = 0;
        }
        public void gatherData(List<Entity> list, EntityFinder finder) {
                if(findMediaAgent){
                        for(int i = 0; i < list.size(); i++){
                                if(list.get(i) instanceof MediaAgent){
                                        m++;
                                        turn = ((MediaAgent)list.get(i)).getTurn(agentID);
                                        pi = ((MediaAgent)list.get(i)).getProb();
                                        movieValues = ((MediaAgent)list.get(i)).getMovieValues()
                                        if(movieseen != -1){
                                                if(movieValues[movieseen] == 1) moviesKnown[movie
                                                else moviesKnown[movieseen] = 0;
                                                movieSaw = movieseen;
                                        }
                                        if(turn == -1 && agentID != -1) System.err.println("Turn
                                        findMediaAgent = false;
                                }
                        }
                }
                if(m == 0) System.err.println("Did not find media agent.");
                if(timeOfDay == 3 && goToWork){
                        tempX = xloc;
                        tempY = yloc;
                        xloc = 1;
                        yloc = 1;
                }
                tempagentMovieValues = agentMovieValues;
                int movieNum = 0;
                double[] marketShares = new double[agentMovieValues.length];
                int recentMovieSeen = 0;
                int i = 0;
                //if it is your turn to see a movie then you find the media agent and see the mov
                if(counter == turn){
                        int m = 0;
                        for(i = 0; i < list.size(); i++){
                                if(list.get(i) instanceof MediaAgent){
                                        m++;
                                        int [] movieValue = new int[agentMovieValues.length];
                                        movieValue = ((MediaAgent)list.get(i)).getMovieValue();
                                        movieNum = ((MediaAgent)list.get(i)).getMovieNum();
                                        marketShares = ((MediaAgent)list.get(i)).getMarketShares(
                                        int movieAssign = ((MediaAgent)list.get(i)).getlastMovieS
                                        recentMovieSeen = ((MediaAgent)list.get(i)).getRecentMovi
                                        decision = new Decision(movieValue, movieNum, marketShare
                                        int movieToSee = decision.makeMixedDecision(prob, movieAs
                                        ((MediaAgent)list.get(i)).seeMovie(movieToSee, movieAssig
                                        i = list.size();
                                }
                        }
                }
                counter++;
                if(counter == agentNum && agentID == -1) reset();
                for(i = 0; i < list.size(); i++){
                        if(list.get(i) instanceof MediaAgent){
                                marketShares = ((MediaAgent)list.get(i)).getActualMarketShares()
                        }
                }
                //Now you search for other agents near you to gather data from
```

```
        for(i = 0; i < list.size(); i++){
                int n = 0;
                if(list.get(i) instanceof Simple2DAgent && list.get(i) != this){
                        if(Math.abs(((Simple2DAgent)list.get(i)).getPosition().getX()-xlo
                                        Math.abs(((Simple2DAgent)list.get(i)).getPositio
                                n++;
                                decision = new Decision(movieValues, movieNum, marketSha
                                int[] otherAgentMoviesKnown = ((Simple2DAgent)list.get(i
                                int temp;
                                for(int p = 0; p < otherAgentMoviesKnown.length; p++){
                                        if(otherAgentMoviesKnown[p] >= 0){
                                                int valueTemp = otherAgentMoviesKnown[p]
                                                int val;
                                                if(valueTemp > 10) val = 1;
                                                else val = 0;
                                                temp = decision.makeQualityDecision(pi, p
                                                if(tempMoviesKnown[p] == -1) tempMoviesKn
                                                if(temp == 1){
                                                        if(valueTemp >= 10 ) tempMoviesKn
                                                        else tempMoviesKnown[p]++;
                                                }
                                                if(temp == 0){
                                                        if(valueTemp >= 10 ) tempMoviesKn
                                                        else tempMoviesKnown[p] = tempMov
                                                }
                                                if(tempMoviesKnown[p] < 0) tempMoviesKnow
                                                if(tempMoviesKnown[p] > 20) tempMoviesKno
                                        }
                                }
                        }
                        if(n == 2)i = list.size();
                }
        checkGoHome(list);
        }
}
public int[] getMoviesKnown(){return moviesKnown;}
private boolean halfChance(){
        if(generator.nextInt(2) == 0) return true;
        else return false;
}
public int getMovieKnown(int i){return moviesKnown[i];}
public int getBestMovie(){return bestMovie;}
public int getMovieSaw(){return movieSaw;}
public Rectangle2D getBoundingSpace() {
        return new Rectangle2D.Double(-20,-20, 20, 20);
```

```
}
public Log getLogData(int i) {return null;}
public List<Entity> getNewlyCreatedEntities() {return new ArrayList<Entity>();}
public Point2D getPosition() {return new  Point2D.Double(xloc,yloc);}
public Renderer getRenderer() {return null;}
public int getSize() {return 0;}
public double searchRadius() {return 150;}
public void setArrayPosition(int pos) {arraypos = pos;  }
public void setID(int x) {agentID = x;  }
public void setLocation(int x, int y) {
        xloc = x;
        yloc = y;
}
public int getID() {return agentID;}
//Location and movie information is updated for the agent
public void update(int timestep) {
        if(xloc == 1 && yloc == 1){
                xloc = tempX;
                yloc = tempY;
        }
        timeStep++;
        moviesKnown = tempMoviesKnown;
        bestMovie = bestMovie(agentMovieValues);
        move();
        if(timeOfDay == 0){
                timeOfDay = 5;
        }
        timeOfDay = timeOfDay - 1;
        agentMovieValues = tempagentMovieValues;
}

private void move(){
                if(checkOutOfBounds()) return;
                else{
                        if(atHome){
                                if(lSideOfStreet){
                                        xloc = xloc + 1;
                                }else{
                                        xloc = xloc - 1;
                                }
                                atHome = false;
                        }
                        else{
                                int num = generator.nextInt(100);
                                if(num > 50){
                                        if(yDirection) yloc = yloc + 1;
                                        else yloc = yloc - 1;
                                }else{
                                        if(num > 20 && num <= 50){
                                                if(yDirection){
                                                        yloc = yloc - 1;
                                                        yDirection = false;
                                                }
                                                else{
                                                        yloc = yloc + 1;
                                                        yDirection = true;
                                                }
                                        }
                                        else{
                                                if(lSideOfStreet){
                                                        xloc = xloc + 1;
                                                        lSideOfStreet = false;
                                                }else{
                                                        xloc = xloc - 1;
                                                        lSideOfStreet = true;
                                                }
                                        }
                                }
                        }
                }
}
private boolean checkOutOfBounds(){
        if(yloc >= lowerBounds && yloc <= upperBounds){
                if(xloc >= lowerXbounds && xloc <= upperXbounds) return false;
                else{
                        if(lSideOfStreet) xloc = lowerXbounds;
                        else xloc = upperXbounds;
                        return true;
                }
        }
        else{
                if(yDirection){
                        yDirection = false;
                        yloc = upperBounds;
                }
                else{
                        yDirection = true;
                        yloc = lowerBounds;
                }
                if(xloc >= lowerXbounds && xloc <= upperXbounds) return true;
                else{
                        if(lSideOfStreet) xloc = lowerXbounds;
                        else xloc = upperXbounds;
                        return true;
                }
        }
}
private void checkGoHome(List<Entity> list){
        double dist = Math.abs(home.getX() - xloc) + Math.abs(home.getY() - yloc);
        if(dist >= timeOfDay) goHome = true;
}
private void goHome(){
        //check if agent is right next to house
        if((Math.abs(home.getX()- xloc) == 1) && (Math.abs(home.getY() - yloc) == 0)){
                if(lSideOfStreet) xloc = xloc -1;
                else xloc = xloc +1;
                atHome = true;
                goHome = false;
                return;
        }
        //if directly accross the street from the house
        if(((buildingNum%2 == 0 && lSideOfStreet != true && xloc == home.getX() - 2) ||
                if(lSideOfStreet){
                        lSideOfStreet = false;
                        xloc = xloc +1;
                }
                else{
                        lSideOfStreet = true;
                        xloc = xloc -1;
                }
        }
        else{
                int n;
                if(((buildingNum%2 == 0 && lSideOfStreet != true) || (buildingNum%2 != 0
                        n = generator.nextInt(100);
                }else n = 0;
                if (n < 70){ // you walk down the street
                        if(home.getY() > yloc) yloc = yloc +1;
                        else yloc = yloc -1;
                }
        }
}
public int bestMovie(int[] marketShares){
        int r;
        int count = 0;
        int num = 0;
        for(int i = 1; i < marketShares.length; i++){
                if(marketShares[i] >= marketShares[num]){
                        if(marketShares[i] > marketShares[num]){
                                num = i;
                                count = 1;
                        }
                        else count++;
                }
        }
        if(count == 1) return num;
        else{
                if(count > 0) r = generator.nextInt(count);
                else return num;
                count = 0;
                for(int i = 0; i < marketShares.length; i++){
                        if(marketShares[i] >= marketShares[num]){
                                if(count == r) return i;
                                else count++;
                        }
                }
        }
        return num;
}
public List<KQML> getMessages() {return new ArrayList<KQML>();}
public void packageData(EntityData ed) {}
public void recieveMessage(KQML message) {}
public void recieveMessages(List<KQML> messages) {}

private int arraypos;
private int xloc;
private int yloc;
private int initialX;
private int initialY;
private int agentID;
private Decision decision;
private int type; //type = 1 means quality decision making, 0 means decision made without
private double prob;
private boolean yourTurn;
private int timeStep = 0;
private int[] agentMovieValues;
private int movieSaw = -1;
private int bestMovie;
private double pi;
private int[] tempagentMovieValues;
```

```
        private int lowerXbounds;
        private int upperXbounds;
        private boolean goToWork;
        private int tempX;
        private int tempY;
        private int[] moviesKnown;
        private boolean findMediaAgent;
        private int turn;
        private int movieseen;
        private int agentNum;
        private Random generator = new Random();

        private int buildingNum;
        private boolean lSideOfStreet; //left side of street
        private boolean atHome = true;
        private boolean goHome = false;
        private boolean yDirection = true; //true = up false = down
        private Point2D home;
        private int lowerBounds;
        private int upperBounds;
        private int timeOfDay = 5;
        private int[] tempMoviesKnown;
        private int counter = 0;
        private int[] movieValues;
        int m = 0;
        private static final long serialVersionUID = 2378493020539472835L;
}
```

Appendix C

Agent Decision Making Class

```
package edu.trinity.cs.mas.informationCascade;

import java.io.Serializable;
import java.util.Random;

public class Decision implements Serializable{

        public Decision(int[] mValue, int mNum, double[] mShares, int rMS, int assignMovie){
                movieValue = mValue;
                movieNum = mNum;
                marketShares = mShares;
                recentMovieSeen = rMS;
                movieAssigned = assignMovie;
        }
        // p is the agents private signal
        // return -1 means no movie was seen
        //used to make decision at the movies
        public int makeDecision(double prob, int movieToSee){
                        boolean p;
                        if(prob >= generator.nextDouble()){
                                        if(movieValue[movieAssigned] == 1)p = true;
                                        else p = false;
                        }else{
                                        if(movieValue[movieAssigned] == 1)p = false;
                                        else p = true;
                        }

                        if(recentMovieSeen == 0){
                                        if(!p) return -1;
                                        else{
```

```
                                        if(halfChance()) return -1;
                                        else return weightedMovie(marketShares,movieToSee
                                }
                }else{
                        if(p) return movieAssigned;
                        else{
                                if(halfChance()) return -1;
                                else return weightedMovie(marketShares,movieToSee);
                        }
                }
}
// pi is equal to the confidence each agent has in the other agents
// return -1 means no movie was seen
// need fixing
public int makeQualityDecision(double pi, int movieToSee, int value, double prob){
                boolean p;
                if(prob >= generator.nextDouble()){
                                if(movieValue[movieAssigned] == 1)p = true;
                                else p = false;
                }else{
                                if(movieValue[movieAssigned] == 1)p = false;
                                else p = true;
                }
                double n = generator.nextDouble();
                if(value == 1){
                        if(n < pi){
                                return 1;
                        }
                        else{
                                if (topN(movieToSee,marketShares)){
                                        if(Double.compare(marketShares[movieToSee],0)==0
                                                if(p)return 1;
                                                else return 0;
                                        }
                                        return 1;
                                }
                                else return 0;
                                }
                }else{
                        if(n < pi){
                                return 0;
                        }
                        else{
                                if (topN(movieToSee,marketShares)){
                                        if(Double.compare(marketShares[movieToSee],0)==0
                                                if(p)return 1;
                                                else return 0;
                                        }
                                        return 1;
                                }
                                else return 0;
                        }
                }
        }
public int makeMixedDecision(double prob, int movieToSee, int[] movieValues){
        boolean p;
        if(prob >= generator.nextDouble()){
                        if(movieValue[movieAssigned] == 1)p = true;
                        else p = false;
        }else{
                        if(movieValue[movieAssigned] == 1)p = false;
                        else p = true;
        }

        if(recentMovieSeen == 0){
                        if(!p) return -1;
                        else{
                                if(halfChance()) return -1;
                                else return bestMovieKnown(movieToSee,movieValues);
                        }
        }else{
                if(p) return movieAssigned;
                else{
                        if(halfChance()) return -1;
                        else return bestMovieKnown(movieToSee,movieValues);
                }
        }
}
public int makeQuickDecision(double prob, int movieToSee, int movieseen, double pi, int[
        boolean p;
        int[] array;
        if(prob >= generator.nextDouble()){
                        if(movieValue[movieAssigned] == 1)p = true;
                        else p = false;
        }else{
                        if(movieValue[movieAssigned] == 1)p = false;
                        else p = true;
        }

        if(movieToSee == -1){
                if(halfChance()) return -1;
                else return topMovie(marketShares,movieToSee);
        }else{
                if(movieseen >= 10){
                        if(generator.nextDouble() < pi) return movieToSee;
                        else{
                                if(p) return movieToSee;
                                else return topMovieChoice(tempMK);
                        }
                }else{
                        if(generator.nextDouble() < pi) return topMovieChoice(tempMK);
                        else{
                                if(p) return movieToSee;
                                else return topMovieChoice(tempMK);
                        }
                }
        }
}
public int makeOriginalQualityDecision(double pi, int movieToSee){
        double n = generator.nextDouble();
        if(recentMovieSeen == 0){
                if(halfChance())return -1;
                else return topMovie(marketShares,movieToSee);
        }else{
                        if(movieValue[movieToSee] == 1){
                        if(n < pi) return movieToSee;
                        else{
                                if(halfChance()) return -1;
                                else return topMovie(marketShares,movieToSee);
                        }
                }else{
                        if(n < pi) return -1;
                        else{
                                if(halfChance()) return -1;
                                else return topMovie(marketShares,movieToSee);
                        }
                }
        }
}
private int weightedMovie (double[] marketShares, int movieToSee){
        if(movieToSee != -1) marketShares = normalizeMarketShares(marketShares, movieToSe
        double prob = generator.nextDouble();
        double counter = 0.0;
        for(int i = 0; i < marketShares.length; i++){
                counter = counter + marketShares[i];
                if(prob < counter) return i;
        }
        return -1;
}
private double[] normalizeMarketShares(double[] marketShares, int movieToSee){
        double[] array = new double[marketShares.length];
        double value = 0.0;
        for(int i = 0; i < marketShares.length; i++){
                if(i != movieToSee){
                        array[i] = marketShares[i];
                        value = value + marketShares[i];
                }else array[i] = 0.0;
        }
        for(int i = 0; i < array.length; i++) array[i] = array[i]/value;
        return array;
}
//returns the highest or one of the highest movies with the highest personal value that i
//otherwise the movie returned is
private int bestMovieKnown(int movieToSee, int[] movieValues){
        int count = 0;
        for(int i = 0; i < movieValues.length; i++){
                if(movieValues[i] >= 10){
                        count++;
                        i = movieValues.length;
                }
        }
        if(count == 0) return bestMovie(marketShares, movieToSee);
        int r;
        count = 0;
        int num = 0;
        for(int i = 1; i < movieValues.length; i++){
                if(i != movieToSee){
                        if(movieValues[i] >= movieValues[num]){
                                if(movieValues[i] > movieValues[num]){
                                        num = i;
                                        count = 1;
                                }
                                else count++;
                        }
                }
        }
        if(count == 1) return num;
        else{
                if(count > 0) r = generator.nextInt(count);
                else return num;
                count = 0;
                for(int i = 0; i < movieValues.length; i++){
                        if(i != movieToSee){
                                if(movieValues[i] >= movieValues[num]){
                                        if(count == r) return i;
                                        else count++;
                                }
                        }
                }
        }
        return num;
}
private int topMovieChoice(int[] marketShares){
        int num = (int)(0.25 * marketShares.length);
        if(num == 0 || num == 1) num = 2;
        int[] selection = new int[num];
        for(int i = 0; i < num; i++){
                selection[i] = 0;
        }
        for(int i = 0; i < marketShares.length; i++){
                int y = smallestSelectionShareInt(selection, marketShares);
                if(marketShares[i] > marketShares[selection[y]]) selection[y] = i;
        }
        int r = generator.nextInt(selection.length);
        return selection[r];
}
//returns one of the movies in the top 25% of marketshares
private int topMovie(double[] marketShares, int movie){
        int num = (int)(0.25 * marketShares.length);
        if(num == 0 || num == 1) num = 2;
        boolean p = true;
        int value = generator.nextInt(marketShares.length);
        while(p){
                value = generator.nextInt(marketShares.length);
                if(value != movie){
                        if(topN(value,marketShares))p=false;
                }
        }
        return value;
}

private boolean topN(int num, double[] marketShares){
        int count = 0;
        for(int i = 0; i < marketShares.length;i++){
                if(marketShares[i] > marketShares[num] && num != i) count++;
        }
        if(count >= 5) return false;
        return true;
}
private int smallestSelectionShareInt(int[] selection, int[] marketShares){
        int r;
        int count = 0;
        int num = 0;
        for(int i = 1; i < selection.length; i++){
                if(marketShares[selection[i]] <= marketShares[selection[num]]){
                        if(marketShares[selection[i]] < marketShares[selection[num]]){
                                num = i;
                                count = 1;
                        }
                        else count++;
                }
        }
        if(count == 1) return num;
        else{
                if(count == 0) return 0;
                else{
                        if(count > 0) r = generator.nextInt(count);
                        else return num;
                        count = 0;
                        for(int i = 0; i < selection.length; i++){
                                if(selection[i] <= selection[num]){
                                        if(count == r) return i;
                                        else count++;
                                }
                        }
                }
        }
        return num;
}
private boolean halfChance(){
        if(generator.nextInt(2) == 0) return true;
        else return false;
}
//returns either the movie with the highest marketshares of one of the movies with the hi
//marketshares
public int bestMovie(double[] marketShares, int movie){
        int r;
        int count = 0;
        int num = 0;
        for(int i = 1; i < marketShares.length; i++){
```

```
                        if(i != movie){
                                if(marketShares[i] >= marketShares[num]){
                                        if(marketShares[i] > marketShares[num]){
                                                num = i;
                                                count = 1;
                                        }
                                        else count++;
                                }
                        }
                }
                if(count == 1) return num;
                else{
                        if(count > 0) r = generator.nextInt(count);
                        else return num;
                        count = 0;
                        for(int i = 0; i < marketShares.length; i++){
                                if(i != movie){
                                        if(marketShares[i] >= marketShares[num]){
                                                if(count == r) return i;
                                                else count++;
                                        }
                                }
                        }
                }
                return num;
        }
        int[] movieValue;
        int movieNum;
        double[] marketShares;
        int recentMovieSeen;
        int movieAssigned;
        private Random generator = new Random();
        private static final long serialVersionUID = 1735466112L;
}
```

Appendix D

Media Agent

```
package edu.trinity.cs.mas.informationCascade;

import java.awt.geom.Point2D;
import java.awt.geom.Rectangle2D;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

import Communication.KQML;
import edu.trinity.cs.mas.Entity;
import edu.trinity.cs.mas.EntityFinder;
import edu.trinity.cs.mas.Renderer;
import edu.trinity.cs.mas.frontend.Log;
import edu.trinity.cs.mas.services.EntityData;

public class MediaAgent implements Entity{
        public MediaAgent(int x, int y, int id,int arrayp, int mNum, double p, int aNum, int[] ag
                xloc = x;
                yloc = y;
                agentID = id;
                movieNum = mNum;
                prob = p;
                agentSeeMovieAssigned = new int[aNum];
                movieAgentSaw = new int[aNum];
                arraypos = arrayp;
                recentMovieSeen = new int[mNum];
                seqMovieNumArray = new int[mNum];
        agentNum = aNum;
        AgentList = agentlist;
        numDiffRuns = numDR;
        if(numDR < 20-1) numPR = numDR-1;
        else numPR = 20-1;
        storeData = new double[5][20-1];
        storeMovieSeen = new int[aNum][numPR];
        storeMarketShares = new double [mNum][numPR];
        reset();
}
private void reset(){
        agentID= agentID - 1;
        assignMovies();
        for(int i = 0; i < movieValues.length; i++){
                System.out.print(movieValues[i]+ ", ");
        }System.out.println();

        movieSeen = new int[movieNum];
        for(int i = 0; i < movieNum; i++){
                movieSeen[i] = 1;
        }
        marketShares = new double[movieNum];
        actualMarketShares = new double[movieNum];
        for(int i = 0; i < agentNum; i++){
                movieAgentSaw[i] = -2;
                agentSeeMovieAssigned[i] = -1;
        }
        nextAgentIndexNum = 0;
        for(int i = 0; i < recentMovieSeen.length; i++){
                recentMovieSeen[i] = 1;
                marketShares[i] = 1 / movieNum;
                actualMarketShares[i] = 0;
                seqMovieNumArray[i] = i + 1;
        }
        seqAgentNumArray = new int[AgentList.length];
        //Shuffle by exchanging each element randomly
        for (int i=0; i<AgentList.length; i++) {
            int randomPosition = generator.nextInt(AgentList.length);
            int temp = AgentList[i];
            AgentList[i] = AgentList[randomPosition];
            AgentList[randomPosition] = temp;
        }
        System.out.println();
        System.out.print("AgentList: ");
        for(int i = 0; i < AgentList.length; i++){
                System.out.print(AgentList[i]+ ", ");
                seqAgentNumArray[i] = i + 1;
        }System.out.println();
        count = 0;
        numGoodMovie = 0;
        counter = 0;
        seeMovieCounter = 0;
        moviesSeen = 0;
        masterCounter = 0;
        lastMovieSaw = generator.nextInt(movieNum);
        firstMovieValue = movieValues[lastMovieSaw];
        firstMovie = lastMovieSaw;
        agentsFound = 0;
}
private void assignMovies(){
        // 1 means High, 0 means Low
        movieValues = new int[movieNum];
        for(int i = 0; i< movieNum; i++){
                if(i<movieNum/2)movieValues[i] = 1;
                else movieValues[i] = 0;
        }
        for (int i=0; i<movieNum; i++) {
            int randomPosition = generator.nextInt(movieNum);
            int temp = movieValues[i];
            movieValues[i] = movieValues[randomPosition];
            movieValues[randomPosition] = temp;
        }
}
public void seeMovie(int movieToSee, int assignedMovie, int agentID){
                seeMovieCounter++;
                if(seeMovieCounter == 3) System.err.println("Agents Found is " + agentsF
                if(movieToSee != assignedMovie){
                        agentSeeMovieAssigned[seeMovieCounter-1] = 0;
                        if(movieToSee != -1){
                                recentMovieSeen[movieToSee] = 1;
                                movieSeen[movieToSee] = movieSeen[movieToSee] + 1;
                        }
                        recentMovieSeen[assignedMovie] = 0;
                }
                else{
                        agentSeeMovieAssigned[seeMovieCounter-1] = 1;
                        movieSeen[movieToSee] = movieSeen[movieToSee] + 1;
                        recentMovieSeen[movieToSee] = 1;
                }
                if(movieToSee != -1){
                        if(movieValues[movieToSee] == 1) numGoodMovie++;
                        moviesSeen++;
                }
                movieAgentSaw[seeMovieCounter-1] = movieToSee;
                proportionGoodMovie = (double)numGoodMovie/(double)(seeMovieCounter);
                pGMSeen = (double)(numGoodMovie)/(double)(moviesSeen);
                System.out.println("seeMovieCounter is " + seeMovieCounter);
                count++;
                updateMovieShare();
                if(movieToSee == -1)lastMovieSaw = assignedMovie;
                else lastMovieSaw = movieToSee;
}
public int getRecentMovieSeen(int num){return recentMovieSeen[num];}
public void displayData(){
                System.out.println("proportionGoodMovie: " + proportionGoodMovie);
                System.out.print("movieValues:  ");
                for(int i = 0; i < movieNum; i++){
                        System.out.print(movieValues[i]+ ",   ");
                }
                System.out.println();
                System.out.print("movieSeen:    ");
                for(int i = 0; i < movieNum; i++){
                        System.out.print(movieSeen[i]+ ",   ");
                }
                System.out.println();
                System.out.print("marketShares: ");
                for(int i = 0; i < movieNum; i++){
                        System.out.print(marketShares[i]+ ", ");
                }
                System.out.println();
                System.out.print("movieAgentSaw: ");
                for(int i = 0; i < movieAgentSaw.length; i++){
                        System.out.print(movieAgentSaw[i]+ ", ");
                }
                System.out.println();
                System.out.print("proportionGoodMovie without -1: " + (double)(numGoodMov
                System.out.println();
                System.out.print("recentMovieSeen: ");
                for(int i = 0; i < recentMovieSeen.length; i++){
                        System.out.print(recentMovieSeen[i]+ ", ");
                }
                System.out.println();
                System.out.println();
}
private void updateMovieShare(){
        for(int i = 0; i < movieSeen.length; i++){
                marketShares[i] = (movieSeen[i])/((double)moviesSeen+20);
                if(moviesSeen == 0) actualMarketShares[i] = 0.0;
                else actualMarketShares[i] = (movieSeen[i]-1)/(double)moviesSeen;
        }
}
public double[] getActualMarketShares(){
        return actualMarketShares;
}
public int getTurn(int agentID){
        agentsFound++;
        for(int i = 0; i < AgentList.length; i++){
                if(AgentList[i] == agentID) return i;
        }
        return -1;
}
public int[] getMovieValue(){return movieValues;}
public int getSpecificMovieValue(int num){return movieValues[num];}
public int getMovieNum(){return movieNum;}
public double[] getMarketShares(){return marketShares;}
public void gatherData(List<Entity> list, EntityFinder finder) {
        masterCounter++;
        System.out.println("masterCounter is " + (masterCounter+1));
        if((masterCounter)%agentNum == 0 && masterCounter != 0){
                System.out.println("End of a Simulation");
                displayData();
                for(int i = 0; i < list.size(); i++){
                        if(list.get(i) instanceof SimpleAgent)((SimpleAgent)list.get(i))
                        if(list.get(i) instanceof Simple2DAgent)((Simple2DAgent)list.get(
                }
                reset();
        }
}
public double getProb(){return prob;}
public Rectangle2D getBoundingSpace() {return new Rectangle2D.Double(-20,-20, 20, 20);}
public Log getLogData(int i) {return null;}
public List<Entity> getNewlyCreatedEntities() {return new ArrayList<Entity>();}
public Point2D getPosition() {return new  Point2D.Double(xloc,yloc);}
public Renderer getRenderer() {return null;}
public int getSize() {return 0;}
public double searchRadius() {return 50;}
public void setArrayPosition(int pos) {arraypos = pos;}
public void setID(int x) {agentID = x;  }
public void setLocation(int x, int y) {
        xloc = x;
        yloc = y;
}
public int getID() {return agentID;}
public int[] getMovieValues(){return movieValues;}
public void packageData(EntityData data) {
}
public void recieveMessage(KQML message) {}
public void recieveMessages(List<KQML> messages) {}
public List<KQML> getMessages() {return new ArrayList<KQML>();}
public void update(int timestep) {counter++;}
public int getlastMovieSaw(){return lastMovieSaw;}

private double[][] storeData;
private int[][] storeMovieSeen;
private double[][] storeMarketShares;
private int[] movieValues;
private double[] actualMarketShares;
private int agentNum;
private double prob;
private int xloc;
private int yloc;
private int agentID;
private int movieNum;
private Random generator = new Random();
private int[] movieSeen;
private int[] recentMovieSeen;
private int count = 0;
private double[] marketShares;
private int[] AgentList;
private int nextAgentIndexNum;
private int arraypos;
private int[] agentSeeMovieAssigned;
private int[] movieAgentSaw;
private double proportionGoodMovie;
private int numGoodMovie = 0;
private int counter = 0;
private int[] seqAgentNumArray;
private int[] seqMovieNumArray;
private int seeMovieCounter = 0;
private int moviesSeen = 0;
private int masterCounter = 0;
private int numDiffRuns;
private int numWrites = 0;
private double pGMSeen = 0;
private int lastMovieSaw;
```

```
        private int firstMovie;
        private int numPR;
        private int agentsFound;
        private int firstMovieValue;
        private final long serialVersionUID = 17354669;
}
```