An Agent-Based Model of Information Diffusion

Final Report

Neža Vodopivec
Applied Mathematics and Scientific Computation Program
nvodopiv@math.umd.edu

Advisor: Dr. Jeffrey Herrmann
Mechanical Engineering Department
jwh2@umd.edu

Abstract: Understanding how information spreads throughout a population can help public health officials improve how they communicate with the public in emergency situations. In this project, I implement a fast, memory-efficient agent-based information diffusion model inspired by the Bass model. I compare my discrete-time simulation to a traditional differential-equation version of the Bass model. Finally, I test my model by seeing how well it describes the real-life spread of information through a Twitter network.

Contents

1. Background

In the weeks following the events of 9/11, seven letters containing dangerous strains of *Bacillus anthracis* were mailed to senators and news agencies. Although the FBI never determined a sender or motive, the attacks informed the country to the possibility of bioterrorism and spurred public health agencies to plan out responses to similar, larger-scale scenarios. Anthrax is not contagious, but its dynamics require a fast dissemination of targeted public health information because newly infected individuals have a far better prognosis when they are treated quickly. To increase effectiveness of a targeted public health message, its broadcasters must understand how information spreads through a population.

Traditionally, information diffusion has been modeled with differential equations that describe the dynamics of a global system — in this case, an entire population. A disadvantage of such models is that they describe only aggregate diffusion patterns, not taking into account that individuals behave in complex ways and that they function within social networks. Thus traditional models can describe the successive increases in the fraction of people who are aware of a given piece of information as a function of time, but they cannot give insight into how this information spreads through space.

Recently, bottom-up modeling in the form of agent-based simulation has gained attention. Agent-based models capture how patterns of behavior at the macro level emerge as the result of the interactions of individuals, or agents, at the micro level. Agent-based models are discrete-time simulations of the interactions in an ensemble of autonomous agents. At each time step, each agent evaluates its situation and makes decisions according to a set of rules.

2. Project Goals

The goal of the current project is to implement an agent-based formulation of the Bass information diffusion model. I create a fast, memory-efficient implementation of this model and use it to simulate how a piece of information spreads through a real-life Twitter network. To ensure that my model is correctly implemented, I compare its output to theoretically predicted results as well as to the output of the traditional differential equation-based Bass model. Finally, I test my model to see how well it describes the actual diffusion of information through Twitter networks and optimize the model's parameters in order to achieve a good fit to such real-world data.

3. Overview of Approach

The Bass model (Bass, 1969) can be used to describe information diffusion. It uses an ordinary differential equation to describe how the fraction of a population aware of a given piece of information increases over time. In an agent-based version of this model (Rand and Rust, 2011), the spread of

information is viewed in the context of a network that has individual agents as decision makers. In my implementation of the agent-based Bass model, I use a network formed from a set of real-world Twitter users, with the observed users as agents. My simulation consists of a series of time steps during which each agent who is still unaware of a given piece of information has an opportunity to change its state. This state change is probabilistic but depends on the fraction of the agent's neighbors who are already aware. Since the simulation is stochastic, I model the spread of information by running the simulation numerous times, then outputting at each time step the mean number of aware agents as well as 95% confidence intervals. (See Section 4.)

I create three implementations of the agent-based Bass model in MATLAB (The MathWorks Inc., 2010). My basic implementation, used as a reference, relies on two adjacency matrices; it determines agents' awareness statuses by recomputing such a status for each node at each iteration. A second version that I refer to as the neighbor-set implementation is more subtle: it stores only edges of the graph and determines agents' statuses by updating each edge exactly once during the entire simulation. My final version – a parallelized neighbor-set implementation – runs multiple simulations simultaneously before computing the mean and confidence intervals. (See Section 5.)

To validate my neighbor-set implementation, I compare samples obtained from many runs of the simulation on a fully connected network, against theoretical distributions. Because the number of aware agents at a given time step depends only on the number aware at the previous time step, the overall system forms a Markov process. I use a transition matrix to compute the distributions of the number of aware agents at each time step and then construct intervals within which 95% of the distribution falls for each time step. I determine, at each time step, the fraction of instances for which the simulation results fall within the theoretically computed interval. If this fraction approaches 95% as the number of simulations increases, this serves as evidence that the model has been properly implemented. (See Section 6.2.)

As another method of code validation, I compare the output of my agent-based Bass model to output generated by the traditional analytical Bass model. If the two models produce nearly identical trajectories when we run the agent-based simulation with a small-enough time step on a fully connected network, we have yet another indicator that the code is correct. I compute the error between the two curves, agent-based and analytical, when the agent-based version is run with successively smaller time steps to test if the agent-based model converges to the analytical curve as Δt decreases. I also examine the correspondence between the parameters in the analytical Bass model to their counterparts in the agent-based model to see if the model trajectories match most closely when identical parameter values are chosen for each model. (See Section 6.3.)

Finally, I test my model against real data to see how well the model describes the actual spread of information through a Twitter network. I take the same Twitter network that was used in the simulations, but this time I record the times that tweets were actually posted by the users. I use these times to compute the aggregate number of users truly aware by various times. I compare the curve of the simulated number of agents aware at each time step to the true curve. I then optimize the model's

parameters by doing a grid search for the values that minimize the error between the two curves. (See Section 7.)

4. The Bass Model

4.1. The Traditional Bass Model

The Bass model was originally developed by a marketer to model brand awareness, but it can also be applied more generally to the diffusion of information. The model is based on the assumption that people get their information from two sources, advertising and word of mouth.

The Bass model describes the fractional change in a population's awareness of a piece of information by the equation:

$$\frac{F'(t)}{1 - F(t)} = p + qF(t)$$

$$F(0) = 0\,,$$

where $F(t)$ is the aware fraction of the population as a function of time, p is the advertising coefficient, and q is the word-of-mouth coefficient. The equation can be interpreted as describing a hazard rate, that is, the conditional probability that a person will become aware of information at time t given that they are not yet aware. In this case, the hazard rate $F'(t)/(1 - F(t))$ is the sum of a constant advertising effect p and a word-of-mouth effect $qF(t)$ that scales linearly in the fraction of population aware. Rearranging terms reveals that the equation is simply the logistic equation in disguise, and has the solution:

$$F(t) = \frac{q - qe^{-(p+q)t}}{q + pe^{-(p+q)t}}.$$

4.2. An Agent-Based Bass Model

We can formulate an agent-based model inspired by the Bass model above. First, we discretize the problem, giving unaware agents an opportunity to become aware of the information at each time step. Then, instead of taking a deterministic aggregate at each time step, we update each agent's state probabilistically. Finally, we consider agents within the context of a social network: instead of allowing each agent to be influenced by the entire population, it is influenced only by its direct neighbors in some underlying directed graph.

The current project focuses on developing an agent-based Bass model that simulates the diffusion of information through Twitter networks. (Twitter is an online service which allows its users to post short

messages, or "tweets", and list which other users they read, or "follow".) In each case, the model's underlying graph Γ has $\mathrm{V}(\Gamma)$ a real-world set of observed Twitter users and $\mathrm{E}(\Gamma)$ the real-world relation "x is 'followed' by y". Two such graphs were used in simulations; they consisted of users who posted messages about the death of Osama Bin Laden and the landfall of Hurricane Irene, respectively. A word-of-mouth transfer of information represents the exchange of information in the form of a Twitter post. The effect of advertising is any external transfer of information, that is, information obtained from a source other than Twitter. We define a Twitter user to be aware when he or she posts a message that conveys the relevant piece of information to followers.

The agent-based Bass model is a discrete-time model in which each agent has one of two states at each time step t: (1) unaware or (2) aware. At the beginning of the simulation, all agents are unaware. At each time step, an unaware agent has an opportunity to become aware. Its state changes with a probability that reflects advertising and word-of-mouth effects. The probability that an agent becomes aware due to word of mouth increases as a function of the fraction of its neighbors who became aware in previous time steps. Once an agent becomes aware, it remains aware for the rest of the simulation.

At each time step, an unaware agent i becomes aware with probability

$$Pr_i(t) = \tilde{p}\Delta t + \tilde{q}\Delta t\frac{a_i(t)}{n_i} - \tilde{p}\tilde{q}\Delta t^2\frac{a_i(t)}{n_i} = \tilde{p}\Delta t + \frac{a_i(t)}{n_i}(1-\tilde{p}\Delta t)\tilde{q}\Delta t\ ,$$

where n_i is the number of neighbors of agent i, $a_i(t)$ is the number of neighbors of agent i that became aware before time t, and $\tilde{p}$ and $\tilde{q}$ are parameters which indicate the effectiveness — per unit time — of advertising and word of mouth, respectively. The first term is the probability that an agent becomes aware due to advertising, the second term that it becomes aware due to word of mouth, and the third term that it becomes aware due to both.

It is not obvious $Pr_i(t)$ represents a true probability, that is to say, we have that $0 \le Pr_i(t) \le 1$. But if we examine its complement

$$1 - Pr_i(t) = 1 - \tilde{p}\Delta t - \frac{a_i(t)}{n_i}(1-\tilde{p}\Delta t)\tilde{q}\Delta t = [1-\tilde{p}\Delta t]\left[1 - \frac{a_i(t)}{n_i}\tilde{q}\Delta t\right],$$

noting that $[1-\tilde{p}\Delta t] \le 1$ and $[1-(a_i(t)/n_i)\tilde{q}\Delta t] \le 1$, we have that $1 - Pr_i(t) \le 1$. Furthermore, because $a_i(t) \le n_i$, we have that $a_i(t)/n_i \le 1$. Therefore, so long as $\tilde{p}\Delta t \le 1$ and $\tilde{q}\Delta t \le 1$, we also have that $[1-\tilde{p}\Delta t] \ge 0$ and $[1-(a_i(t)/n_i)\tilde{q}\Delta t] \ge 0$, so that $1 - Pr_i(t) \ge 0$. Combining inequalities, then, we have that $0 \le 1 - Pr_i(t) \le 1$, and so $0 \le Pr_i(t) \le 1$ as well.

5. Implementation of the Agent-Based Bass Model

5.1. Approach

In their paper, "Agent-Based Models of Information Diffusion", Auzolle and Herrmann (2012) describe their implementations of four types of agent-based diffusion simulations. Their codebase, written in NetLogo, a programming language used to develop agent-based simulations (Tisue and Wilensky, 2004), turned out not to be fast enough to handle large networks. The goal of the current project is to code the agent-based Bass model in MATLAB with the hope of producing a faster, more memory-efficient implementation. I implement three versions of this model. First, I code a basic implementation to use as a reference. Then, I implement the model using a more efficient updating rule and taking advantage of sparse data structures. I call this second implementation the neighbor-set implementation. Finally, I code a parallelized version of the neighbor-set implementation.

5.2. Basic implementation

The basic implementation depends on the use of an adjacency matrix to store relationships between agents and to record agents' awareness statuses. A straightforward algorithmic description of the basic simulation is as follows.

5.2.1 Algorithm (Basic)

Arbitrarily identify the N agents of the graph Γ with the set $1, \ldots, N$. Let E denote the $|\mathrm{E}(\Gamma)| \times 2$ matrix listing all (directed) edges of Γ as ordered pairs of nodes.

INPUT: matrix E, parameters $\tilde{p}$ and $\tilde{q}$, parameter Δt.

1. Create an $N \times 1$ bit vector X, initialized to $\mathbf{0}$. X will keep track of which nodes are aware.
2. Create an $N \times 1$ bit vector ΔX, initialized to $\mathbf{0}$. ΔX will keep track of which nodes are newly aware.
3. Create an $N \times N$ adjacency matrix A, initialized to $\mathbf{0}$. Set $A(i, j)$ to 1 if the vector (i, j) appears as a row of E. A will remain static throughout the simulation.
4. Create an $N \times N$ adjacency matrix B, initialized to $\mathbf{0}$. B will keep track of the directed edges in A whose upstream node is aware, as marked in S.
5. At each time step:
   a. For each node i:
      i. Check $X(i)$ to determine whether node i is already aware. If so, skip it.
      ii. With probability $\tilde{p}\Delta t$, make node i newly aware by setting $\Delta X(i)$ to 1.
      iii. Look up node i's upstream neighbors in $A(*, i)$, and then its aware upstream neighbors in $B(*, i)$. Determine what fraction f of upstream neighbors are aware. With probability $f \times \tilde{q}\Delta t$, make node i newly aware by setting $\Delta X(i)$ to 1.
   b. Once all nodes have been processed:

i. Record the newly aware nodes marked in ΔX as aware in X.
ii. For each newly aware node marked in ΔX, copy the corresponding row of A to the corresponding row of B.
iii. Reset ΔX to $\mathbf{0}$.

6. Stop once all nodes have become aware, or after a maximum number of iterations.

OUTPUT: complete history of the bit vector X.

5.3 Neighbor-Set Implementation

The neighbor-set implementation benefits from a more efficient updating rule and from custom sparse data structures tailored to this new updating rule. The result is a faster runtime and more efficient use of memory.

5.3.1 Formulation

In order to decide whether to change the status of an unaware node, the node's number of unaware upstream nodes (its "awareness number") must be computed. The basic implementation effectively recomputes each node's awareness number from scratch at every time step. But because changes in the awareness number are entirely due to nodes which have just become aware, such a computation seems wasteful. This suggests a possible improvement: a preliminary pass through just the newly aware nodes which updates just their downstream nodes. After this preliminary step, we can proceed as in the basic implementation, but without needing to recompute awareness numbers.

This new updating procedure suggests a further possible improvement: replacing the network's adjacency matrix with a sparse data structure which reflects the structure of the updating rule. Information about adjacency can be stored by rewriting the adjacency relation as a function $f: \mathrm{V}(\Gamma) \to 2^{\mathrm{V}(\Gamma)}$ which returns a node's downstream nodes. Concretely, this function is most naturally implemented as a vector of length $|E|$ concatenating the output sets of f together with a list of pointers marking the start of each set. Note that the coding of this function can also be thought of as an $|E| \times 2$ ordered list of the coordinates of the nonzero entries in the original adjacency matrix.

5.3.2 Algorithm (Neighbor-Set)

Arbitrarily identify the N agents of the graph Γ with the set $1, \ldots, N$. Let E denote the $|\mathrm{E}(\Gamma)| \times 2$ matrix listing all (directed) edges of Γ as ordered pairs of nodes.

INPUT: matrix E, parameters $\tilde{p}$ and $\tilde{q}$, parameter Δt.

1. Create an $N \times 1$ bit vector X, initialized to $\mathbf{0}$. X will keep track of which nodes are aware.

2. Create three $N \times 1$ vectors n, a, and Δa, all initialized to $\mathbf{0}$. These vectors will list, for each node, a count of the node's (upstream) neighbors, the node's aware (upstream) neighbors, and the node's newly aware (upstream) neighbors, respectively.

3. Passing through the rows of E, note when k appears as the second entry of the row, and increment $n(k)$. Because $n(k)$ will have been incremented once for each time k appeared in E paired with an (upstream) neighbor, it will count k's (upstream) neighbors.

4. Ensure that the rows of E are sorted in lexicographic order. This will guarantee that all the edges out of a given node appear consecutively within E.

5. By noting the rows at which the entries of $E(*,1)$ jump, create an $N \times 1$ vector I_* whose kth entry is the starting index (in E) of the consecutive run of all edges out of node k. In the same way, create a matching $N \times 1$ vector I^* for the ending indices.

6. $E(*,2)$ may now be viewed as a node-by-node concatenation of each node's (downstream) "neighbor set", with pointers to the location of the kth set given by $I_*(k)$ and $I^*(k)$. To reinforce this view, rename $E(*,2)$ to S and then discard E.

The quantities n, I_*, I^*, and S constitute a convenient encoding of Γ, and now that they have been computed they will remain static. At each time step, we will be updating X after examining these quantities and the dynamic variables a and Δa.

7. At each time step:

    a. For each node i:

        i. Check $X(i)$ to determine whether node i is already aware. If so, skip it.

        ii. With probability $\tilde{p}\Delta t$, make node i newly aware by setting $X(i)$ to 1.

        iii. Look up the count of node i's upstream neighbors in $n(i)$, and then the count of its aware upstream neighbors in $a(i)$. Determine what fraction f of upstream neighbors are aware.

        iv. With probability $f \times \tilde{q}\Delta t$, make node i newly aware by setting $X(i)$ to 1.

        v. If node i is now aware, then all of its downstream neighbors now have a newly aware upstream neighbor. Look up node i's downstream neighbors in the entries of S that lie between indices $I_*(i)$ and $I^*(i)$. For each downstream neighbor, increment the corresponding entry of Δa.

    b. Once all nodes have been processed:

        i. Increment a by Δa. (The computation inside the loop is using the entries of a, and so we must save making increments to a for a final step occurring after the loop.)

ii. Reset Δa to $\mathbf{0}$.

8. Stop once all nodes have become aware, or after a maximum number of iterations.

OUTPUT: complete history of the bit vector X.

5.3.3 Parallel Implementation

We can increase the efficiency of the neighbor-set implementation further through parallelization**.** We model information diffusion by running the simulation numerous times and then computing the mean and 95% confidence intervals at each time step. This process lends itself naturally to parallelization. The neighbor-set implementation was parallelized so that multiple simulations run simultaneously. The results of each simulation are logged at the end of each run and the complete set of data is analyzed after all runs have been completed. The parallelization was implemented using MATLAB's `parfor` command.

6. Validation of the Agent-Based Bass Model

6.1 Mutual Validation among Four Implementations

The simulation was run once using the the NetLogo implementation, the basic implementation, and both versions of the neighbor-set implementation, respectively. The outputs of the four implementations were compared by computing the L^1 error between each pair of curves. When run with the same random numbers, the basic implementation and the two neighbor-set implementations produced identical results (a zero error). There was no easy way to induce identical behavior in the existing NetLogo code, but its output broadly matches the other implementations. The curves for each of the serial implementations run on four datasets are plotted below.

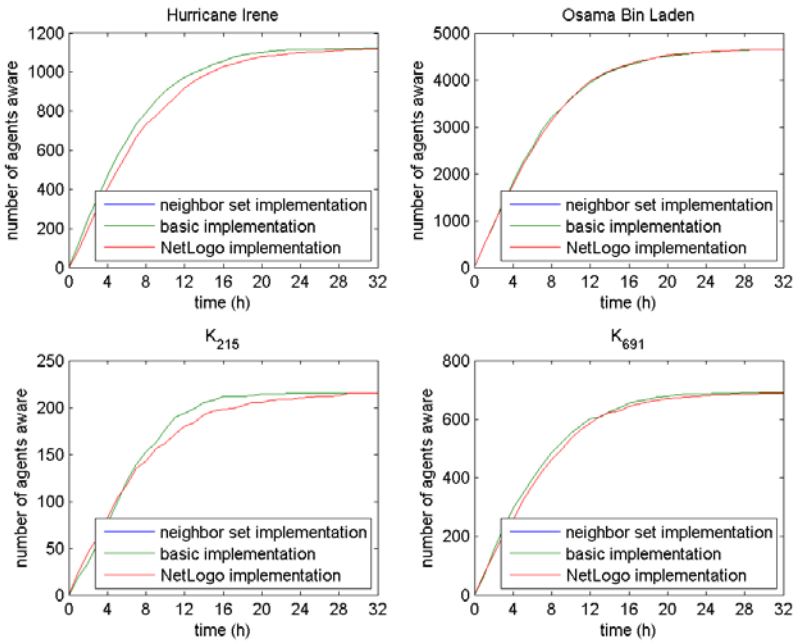


Figure 1: Implementation Comparison

6.2 Comparing Simulation Output to Theoretical Distributions

A second way to validate the simulation would be to compare samples, obtained from many runs of the simulation, against theoretical distributions. In general, determining the theoretical distribution of a statistic taken from the model is an interesting but very difficult question. The question becomes easier when we make a simplifying assumption: all agents are connected. (To make the bookkeeping slightly easier, we will even assume that each agent is connected to itself.) As a result, local network structure disappears and, for each agent i, $a_i(t)/n_i$ is simply $A(t)/N$, the aware fraction of the network. Then, at each time step, the probability that an unaware agent becomes aware does not depend on i:

$$Pr(t) = \tilde{p}\Delta t + \tilde{q}\Delta t\frac{A(t)}{N} - \tilde{p}\tilde{q}\Delta t^2\frac{A(t)}{N} = \tilde{p}\Delta t + \frac{A(t)}{N}(1-\tilde{p}\Delta t)\tilde{q}\Delta t\,.$$

At every time step, each agent is in one of two states: aware or unaware. Thus, there are 2^N possible states for the system. The number of aware agents is not binomially distributed at each time step: unless $\tilde{q} = 0$, the awareness of an agent correlates to the past, and therefore present, awareness of other agents. Nevertheless, because its current state depends only on its previous state, the overall system forms a Markov process. The map which counts the number of aware agents takes the system's state space to a reduced space with only $N + 1$ states. Because of the very special structure of the system, the counting map respects the original Markov process, giving rise to a compatible Markov process on the reduced state space. Rather than describe the first Markov process and then trace

through the counting map to obtain a description of the second Markov process, it is easier to describe the second Markov process directly.

6.2.1 Computing Theoretical Distributions

Let X_t be a random variable giving the number of agents aware at time step t. Since the number of agents aware at a given time step depends only on the number aware at the previous time step, the sequence $X_0, X_{\Delta t}, X_{2\Delta t}, \ldots$ forms a Markov chain. We can represent its transition probabilities with a matrix T.

Because the state space consists of the $N+1$ elements $0,1,\ldots,N$, it is convenient to index the rows and columns of the transition matrix T from 0 instead of 1. The system is in state j if exactly j agents are aware. The probability $T(i,j) = \Pr(X_{t+\Delta t} = i | X_t = j)$ of transitioning from state j to state i at time step $t + \Delta t$ is:

$$T(i,j) = \begin{cases} \binom{N-j}{i-j}(P+jQ)^{i-j}(1-P-jQ)^{N-i}, & i \geq j \\ 0, & i < j \end{cases},$$

where $P \coloneqq \tilde{p}\Delta t$ and $Q \coloneqq N^{-1}(1-\tilde{p}\Delta t)\tilde{q}\Delta t$.

6.2.2 Constructing Theoretical Intervals

We can use the transition matrix T to compute the distributions of the random variables X_t. With these distributions in hand, we wish to determine at each time step t an interval $[a_t, b_t]$ such that $\Pr(a_t \leq X_t \leq b_t) = 0.95$. But since our distributions are discrete, such an interval will not in general exist. We therefore choose the a_t which minimizes $|\Pr(X_t < a_t) - 0.025|$ and, similarly, the b_t which minimizes $|\Pr(X_t > b_t) - 0.025|$. The probability that X_t falls within the interval, we call P_t. P_t is approximately but not exactly 95%, and the P_t-intervals we compute below play the role of 95% intervals for a discrete distribution.

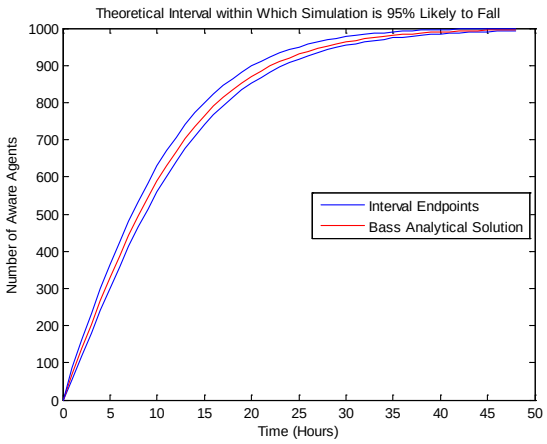


Figure 2: P_t-intervals

Figure 4 shows curves for a_t (bottom) and b_t (top) at each time step t, such that $\Pr(a_t \leq X_t \leq b_t) \approx 0.95$. Between the curves lies the analytical solution to the Bass ODE.

6.2.3 Testing Theoretical Intervals on the Simulation

At each time step, does our simulation produce a number of aware agents whose distribution matches the theoretical distribution? One obvious way to make a comparison is to run the simulation numerous times and determine, at each time step, the fraction of instances for which the simulation results fall within the theoretically computed P_t-interval. If that fraction is indeed P_t, we have some validation that the simulation is correctly implemented.

After running the simulation n times, we compute at each time step t an empirical fraction $S_{n,t}$ of instances for which the simulation results fall within the P_t-interval. We would like to know how well $S_{n,t}$ compares to P_t as n increases.

$n = 5{,}000$ $n = 50{,}000$

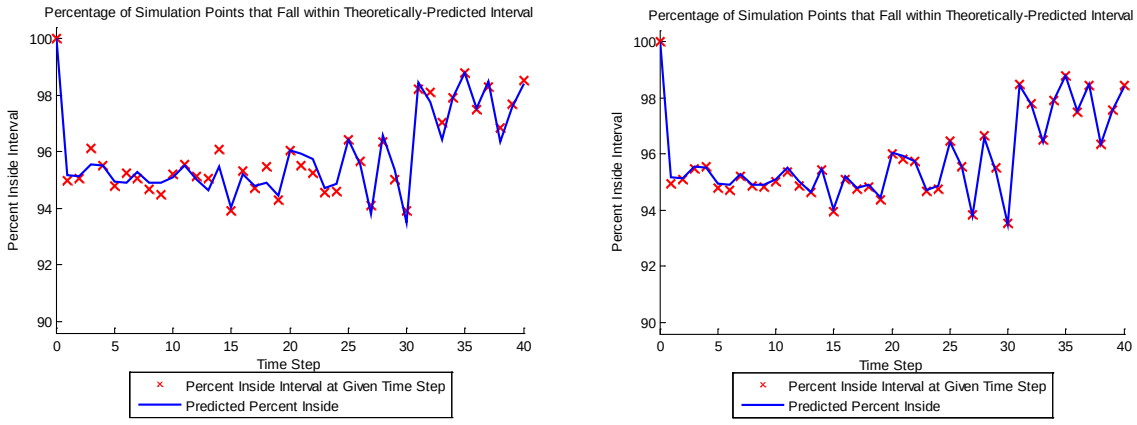


Figure 3: Percentage of Simulation Points that Fall within Theoretically Predicted Intervals

For each n and t, we can compute $E_{n,t} = \left|P_t - S_{n,t}\right|$ the discrepancy between the fraction of times that X_t is predicted to fall in the interval $[a_t, b_t]$ and the fraction of times that it actually does. To summarize the behavior of the $E_{n,t}$ for a given n, we can take their mean $\bar{E}_n$ over t. In addition to computing the first moment over t, we also compute the second moment σ_n^2. The plots below show the behavior of $\bar{E}_n$ and σ_n^2 as n increases.

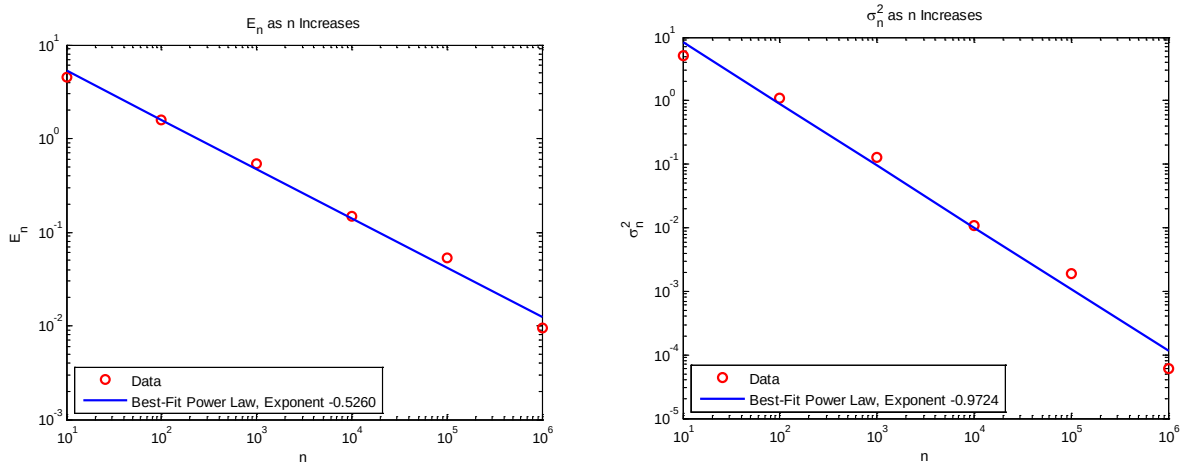


Figure 4: $\bar{E}_n$ and σ_n^2 as n Increases

6.3 Comparing the Agent-Based Bass Model to the Analytical Bass Model

Another way to validate the implementation of the agent-based Bass model is by comparing it to the analytical Bass model. We again restrict ourselves to the case of a fully connected network. Each time we run the simulation, we can compute the total fraction $\Phi(t)$ of the network that has become aware as a function of time. We can run the simulation numerous times and recompute $\Phi(t)$ multiple times to obtain an average $\bar{\Phi}(t)$. In this section, we compare $\bar{\Phi}(t)$ to $F(t) = \frac{q - qe^{-(p+q)t}}{q + pe^{-(p+q)t}}$, the cumulative aware fraction of the network given by the analytical Bass ODE.

6.3.1 Convergence of Agent-Based Model to Analytical Model

We would like to know if the agent-based model converges to analytical curve as Δt decreases. Beginning with $\Delta t = 1$ hour and successively decreasing time-step length, we compute the L^1 error between $\bar{\Phi}(t)$ and $F(t)$ for each value of Δt. As shown in figure 7, $\bar{\Phi}(t)$ converges to $F(t)$ – evidence that the agent-based model was correctly implemented.

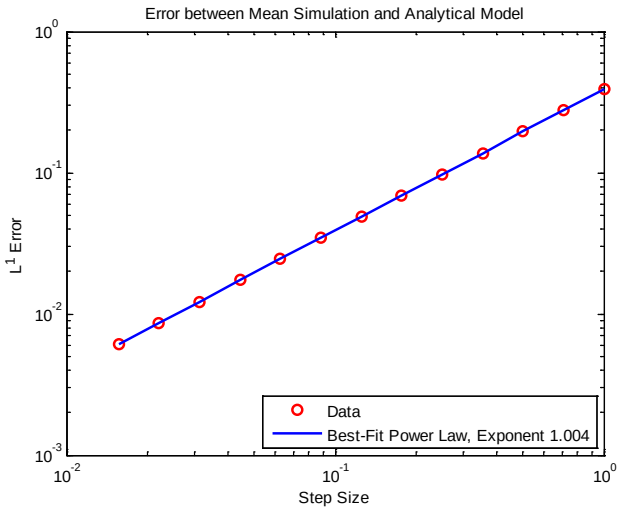


Figure 5: L^1 Error Between $\overline{\Phi}(t)$ and $F(t)$

6.3.2 Comparing the Parameters in the Two Models

Another way to explore the relationship between the analytical and agent-based models is to determine the correspondence between parameters p and q in the analytical Bass model to their counterparts $\tilde{p}$ and $\tilde{q}$ in the agent-based model. We would expect that the model trajectories would be most similar when $\tilde{p} = p$ and $\tilde{q} = q$. To test this, we fix the values of the parameters of the analytical model at $p = 0.07$ and $q = 0.06$. and perform a grid search. For each of an exhaustive selection of $(\tilde{p}, \tilde{q})$ pairs, we run the simulation 100 times and take the mean pointwise in time. We then choose the $(\tilde{p}, \tilde{q})$ pair $(\tilde{p}^*, \tilde{q}^*)$ that minimizes the L^1 error between the analytical Bass curve with parameters $p = 0.07$ and $q = 0.06$ and the output of the agent-based model with values of $\tilde{p}$ and $\tilde{q}$.

The color map gives, as a function of $\tilde{p}$ and $\tilde{q}$, the L^1 error between the agent-based curves with $(\tilde{p}, \tilde{q})$ and analytical curve with $(p, q) = (0.06, 0.07)$. The minimum error occurred at $(\tilde{p}^*, \tilde{q}^*) = (0.0690\ , 0.0595)$.

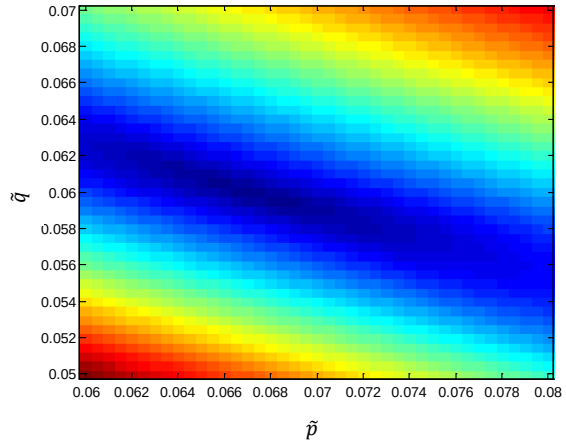


Figure 6*:* L^1 Error as a Function of $\tilde{p}$ and $\tilde{q}$

We can compute the best-fit quadratic approximation to the error function. Figure 8 shows the contours of this quadratic approximation. The conic center of the approximation is too far to the right to match the parameters $(\tilde{p}^*, \tilde{q}^*) = (0.0690, 0.0595)$, that mimimize the error.

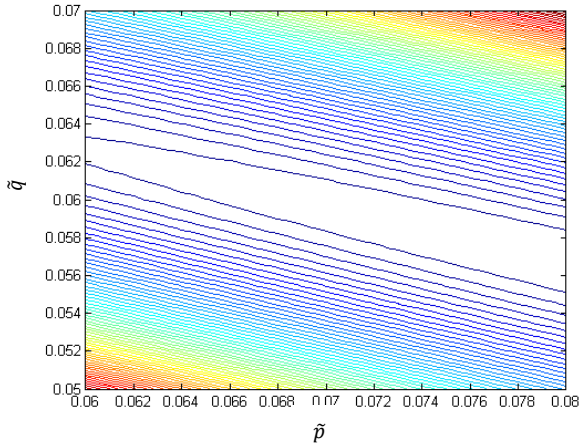


Figure 7: Best L^2 Fit to Error

$$E = 9871[\ 0.2832(\tilde{p} - 0.09041) + 0.9591(\tilde{q} - 0.05368)]^2$$
$$+ \ 22.67[-0.9591(\tilde{p} - 0.09041) + 0.2832(\tilde{q} - 0.05368)]^2$$
$$+ 0.2438\ .$$

7. Results

7.1 Simulation Output

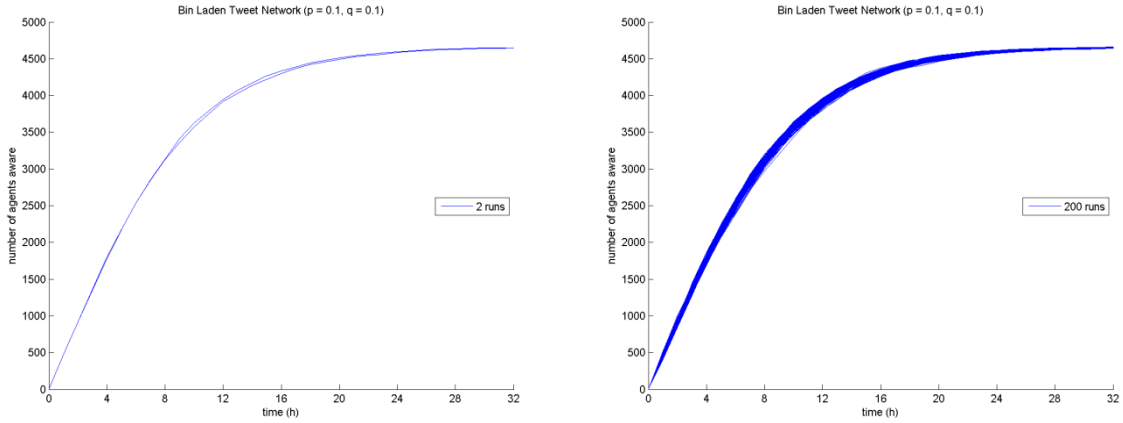


Figure 8: Simulation Trajectories on the 'Bin Laden' Twitter Network

The plots above show the number of agents aware at each time step of the simulation. Figure 1a shows the trajectories for two simulation executions; figure 1b shows trajectories for 200 executions. Since the simulation is stochastic, a single run provides, at each time step, only a sample of the network's true behavior — the trajectories are close but not the same.

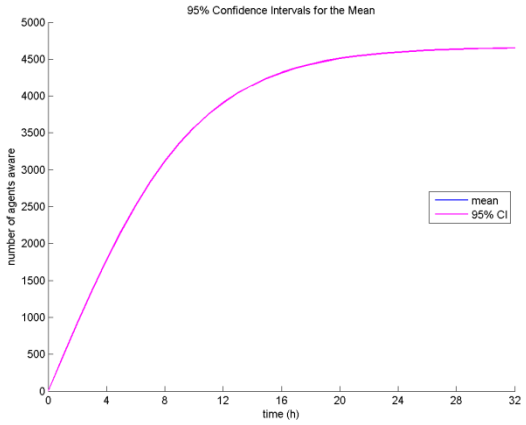


Figure 9: Confidence Intervals Surrounding the Simulation Mean at Each Time Step

The simulation was run 100 times and the mean and 95% confidence intervals were computed.

7.2 Implementation Time and Memory Requirements

Table 1: Time and Space Efficiency of Model Implementations

	Bin Laden Network (7.27 MB, 4.7K nodes, 477K edges)		
Implementation Type	**Memory**	**Simulation Time**	**1K Simulations' Time**
NetLogo	—	~3 min.	—
Basic	506.73 MB	13.8 sec.	3.6 hr.
Neighbor-Set, Serial	18.55 MB	0.8 sec.	12.5 min.
Neighbor-Set, Parallel	18.55 MB	10.2 sec.	8.6 min.

	Hurricane Irene Network (0.70 MB, 1.1K nodes, 46K edges)		
Implementation Type	**Memory**	**Simulation Time**	**1K Simulations' Time**
NetLogo	—	~50 sec.	—
Basic	30.31 MB	3.74 sec.	52.2 min.
Neighbor-Set, Serial	1.84 MB	0.27 sec.	3.6 min.
Neighbor-Set, Parallel	1.84 MB	9.46 sec.	2.6 min.

Table 1 gives the time and space requirements for the NetLogo, basic, neighbor-set serial, and neighbor-set parallelized implementations of the agent-based Bass model for the two Twitter networks. The neighbor-set implementations stand out as the fastest and the most memory-efficient. Memory requirements for each implementation were computed by summing the number of bytes used to store all the variables created during one run of the simulation. Times for the parallelized neighbor-set implementation were obtained by running two simulations in parallel and they include MATLAB's overhead time for initiating parallelization. These times indicate that using the parallelized code is effective only when repeating the simulation many times. Values listed for the NetLogo implementation are an estimate as the program is designed to be run in two stages; the table gives runtimes for the second stage. Six out of the ten times it was executed, the NetLogo program was stopped after running for longer than an hour. The best runtime (out of ten) was taken for each dataset.

7.3 Testing the Model on Real-World Twitter Data

We wish to determine how well the Bass agent-based model describes the actual spread of information through a Twitter network. One way to do this is to analyze the actual "tweets" made by the users in the Twitter networks we used for the simulations — those posting the news of Osama Bin Laden's death and Hurricane Irene's landfall, respectively. We can record the first time that each user posted a message containing a relevant string of text, take this time as the definition of his or her "true awareness time", and then integrate to obtain the aggregate number of users aware by various times. Now, we can run our simulation, compute the curve representing the number of agents aware at each time step, and compare the computed curve to the true curve.

7.3.1 Optimizing the Parameters

Our model has two free parameters, $\tilde{p}$ and $\tilde{q}$. We can use a grid search to find the values of $\tilde{p}$ and $\tilde{q}$ for which the simulation most closely approximates the observed Twitter data. For each of an exhaustive selection of $(\tilde{p}, \tilde{q})$ pairs, we run the simulation 100 times and take the mean pointwise in time. We then choose the $(\tilde{p}, \tilde{q})$ pair $(\tilde{p}^*, \tilde{q}^*)$ that minimizes the L^1 error between the simulation means and the real data. For the Bin Laden dataset, a color map below shows the error as a function of $\tilde{p}$ and $\tilde{q}$ when $\Delta t = 1$ hour. The optimal parameters were: $\tilde{p}^* = 0.099, \tilde{q}^* = 0.001$

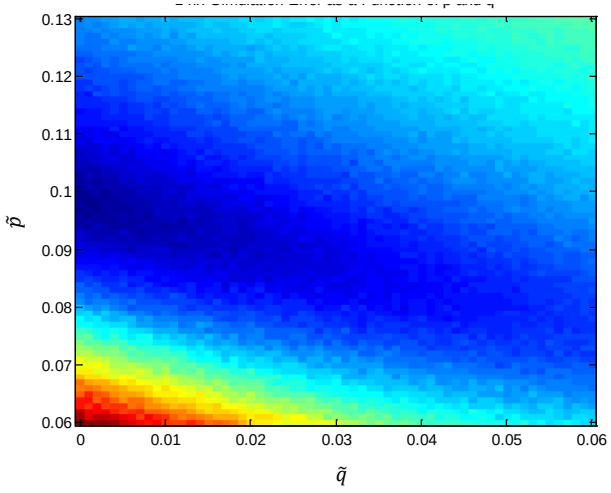
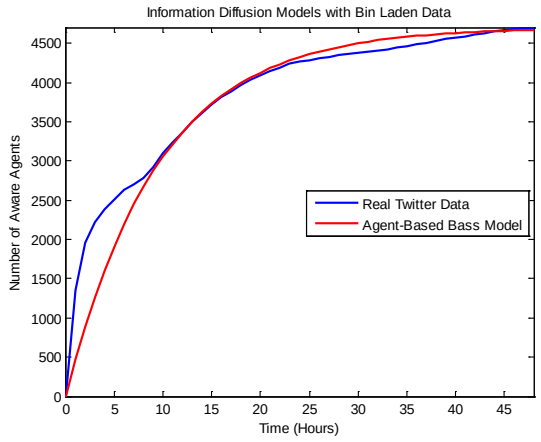
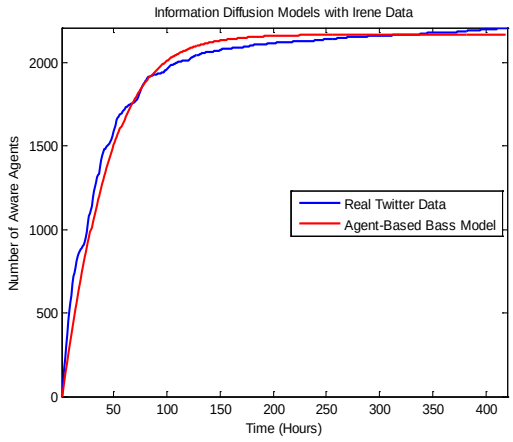


Figure 10: Error as a Function of $\tilde{p}$ and $\tilde{q}$ for $\Delta t = 1$

The plots below compare the true number of aware Twitter users to the number of aware agents as given by the optimal agent-based model. Figure 3a shows a plot for the Bin Laden dataset, and Figure 3b shows a plot for the Irene dataset. For each dataset, the simulation values plotted were obtained by averaging 100 runs.



$\tilde{p}$ = 0.099, $\tilde{q}$ = 0.001.



$\tilde{p}$ = 0.023, $\tilde{q}$ = 0.001

Figure 11: Comparing the Agent-Based Model to Observed Twitter Data

7.3.2 Error as a Function of the Time Step

We would like to know if the optimal parameters and the error are affected by our choice of Δt. In addition to the highly visible parameters $\tilde{p}$ and $\tilde{q}$, and the underlying graph Γ, the agent-based Bass model also depends on the rather less visible parameter Δt. It would be nice to know whether refining Δt substantially affects the model's fit to real-world data. To investigate this question, the search for optimal parameters $(\tilde{p}^*, \tilde{q}^*)$ for the Bin Laden dataset was conducted anew using $\Delta t = 0.25, 0.50, 0.75$ hours, in exactly the same way as with $\Delta t = 1$ hour. Table 2 below shows the optimal parameters for each Δt, along with the corresponding minimal L^1 error. Using different values of Δt does not significantly affect how well the model fits the real-world data.

Table 2: Optimal Parameters and Error as Δt Changes

Δt (hr)	Optimal $\tilde{p}$	Optimal $\tilde{p}\Delta t$	Optimal $\tilde{q}$	Optimal $\tilde{q}\Delta t$	Error
1	0.099	0.099	0.0010	0.0010	7.4632e+003
0.75	0.101	0.076	0.0008	0.0006	7.5229e+003
0.5	0.102	0.052	0.0009	0.0005	7.4734e+003
0.25	0.103	0.026	0.0010	0.0003	7.5602e+003

8. Project Milestones

My milestones as described in my proposal:

October	Develop basic simulation code. Develop code for statistical analysis of results.
November	Validate simulation code by checking corner cases, sampled cases, and by relative testing. Validate code against analytical model.
December	Validate simulation against existing NetLogo implementation. Prepare mid-year presentation and report.
January	Investigate efficiency improvements to code. Incorporate sparse data structures.
February	Parallelize code. Test code efficiency against existing NetLogo implementation.
March	Test model against empirical Twitter data. Create visualization of model, time permitting.
April	Write final project report and prepare presentation.

My milestones as achieved:

October	Developed basic simulation code. Wrote a more efficient neighbor-set implementation of simulation.
November	Validated code against analytic model. Developed code for statistical analysis of results. Analyzed confidence intervals.
December	Validated simulation against existing NetLogo implementation. Prepared mid-year presentation and report.
January	Tested model against empirical Twitter data. Performed parameter searches to determine best fit. Tested best fit as a function of Δt.
February	Parallelized code. Analyzed convergence of agent-based model to analytical model. Computed probability distributions using Markov chains.
March	Computed and tested 95% intervals. Compared correspondence of parameters in agent-based model to those in analytical model. Analyzed the error.
April	Revised and re-tested ~95% intervals. Fitted quadratic function to error. Wrote final project report and prepared presentation.

9. Deliverables

I will submit all deliverables promised in my proposal (1-4) as well as a few additional ones (5-7).

1. Code for my simulation.
2. Code for my statistical analysis.
3. A plot showing at each time step the mean and both ends of a 95% confidence interval based on data collected from numerous runs of the simulation.
4. A comparison of my code's running time against that of the existing NetLogo implementation.
5. Faster neighbor-set implementation of the simulation, parallelized.
6. Code for grid search to determine best-fit parameters.

7. Code to compute and test theoretical distributions for fully connected networks using Markov chains.

10. Conclusion

The current project was an effort to implement an agent-based Bass information diffusion model. Using an efficient updating rule and taking advantage of sparse data structures produced a more time- and memory-efficient code. The current implementation is faster and runs more reliably than a previous NetLogo implementation, allowing simulations to be performed on larger, denser networks in future research.

With the correct parameters, the agent-based Bass model provides a reasonable description of real-world Twitter information diffusion. The model parameters that produce the best fit are specific to each data set. Varying the length of time steps has little effect on the fit between the model and real data.

We can compare the discrete agent-based model to the analytical Bass model. As time steps decrease in length, the agent-based Bass model converges to the analytical Bass model. Parameters p and q of the analytical model correspond well to parameters $\tilde{p}$ and $\tilde{q}$ of the agent-based model.

11. References

Auzolle, Ardechir and Herrmann, Jeffrey (2012). "Agent-Based Models of Information Diffusion". Working paper, University of Maryland, College Park, Maryland.

Bass, Frank (1969). "A new product growth model for consumer durables". *Management Science* 15 (5): p. 215–227.

Chandrasekaran, Deepa and Tellis, Gerard J. (2007). "A Critical Review of Marketing Research on Diffusion of New Products". *Review of Marketing Research*, p. 39-80; Marshall School of Business Working Paper No. MKT 01-08.

Devore, J.L. (1991). *Probability and statistics for engineering and science*. Brooks/Cole.

Dodds, P.S. and Watts, D.J. (2004). "Universal behavior in a generalized model of contagion". *Phys. Rev. Lett*. 92, 218701.

Karlin, S. and Taylor, H.M. (1968). *A first course in stochastic processes*. Academic Press.

MATLAB v. 7.10.0. (2010). Natick, Massachusetts: The MathWorks Inc.

Mahajan, Vijay, Muller, Eitan and Bass, Frank (1995). "Diffusion of new products: Empirical generalizations and managerial uses". *Marketing Science* 14 (3): G79–G88.

Rand, William M. and Rust, Roland T. (2011). "Agent-Based Modeling in Marketing: Guidelines for Rigor (June 10, 2011)". *International Journal of Research in Marketing*; Robert H. Smith School Research Paper No. RHS 06-132.

Tisue, S. and Wilensky, U. (2004). *NetLogo: A simple environment for modeling complexity*. Paper presented at the Fifth Proceedings of the International Conference on Complex Systems, Boston.