Cascades, Product Conversion, Whatever

T. Martian & E. Brinkbot

December 11, 2012

1 Introduction

The study of network games is a relatively new and very active area of research. It has become both more important and more applicable since the internet has made large scale social interaction easier and more quantifiable. A common topic in network games is that of contagious or cascading processes, in which a number of agents start with some property they then spread to their neighbors under some spreading rules. This model naturally represents phenomena such as the spreading of trends, technologies, or influence through people or groups, or cascading failures in structures such as power grids or banks. Our proposed research is about theoretical models of spreading and cascading processes on networks.

Many models with simple spreading rules have been proposed [1, 4, 7] to explain, for example, how breaking news spreads over the internet or how a new technology (such as the iPod) spreads in popularity. These spreading rules usually take one of two forms. The rules either govern a stochastic spreading process, or they assume that each node in the network is a strategic agent which makes a decision (e.g. to buy an iPod or to buy a Zune) and derives greater utility if his neighbors make the same decision. To the best of our knowledge, all current strategic agent models make a simplifying assumption that the agents are only myopically strategic [2]: they make decisions at each round to maximize their utility at that point in the game. But, each agent's behavior can have profound impact on the future state of the network and thus the agent's own final utility. Our interest is in studying network processes under the more realistic assumption that agents make decisions conscious of their future influence on outcomes, in order to maximize their expected final utility.

More specifically, we propose to research the impact of having strategic users: how much different are results if we assume users are strategic instead of myopic. To answer this question, we will use a model introduced in [2] and described in Section 2. Our contribution is to fully characterize the behavior of this model on a specific class of graphs, the 2-blockmodels, in hopes of learning more about the general case. Blockmodels have been studied extensively in the past [6, 5] as a natural framing of networks in which we can think of a node having a type which influences its characteristics. For example, a blockmodel describing a social network could have types for each grade, and students with each type are more likely to connect to their own grade, and unlikely to connect to grades very far away.

The analysis of strategic behavior on networks is not a new concept. Kearns et. al. introduced a convenient method for expressing games of this sort with their seminal work on graphical games [3]. But, to the extent of our knowledge, nobody has attempted to characterize full graphical game models for cascade spreading. This is what we do below.

2 Model

We propose a cascade spreading model that borrows heavily from the scheduler mediated cascade process introduced in [2]. In our model every node in an undirected graph is a risk neutral rational agent. Each node is also assigned a binary type which affects their utility, pulled from a binomial distribution. In this paper we call the types Yes (Y) and No (N). There is an additional scheduler agent, which, at each round of the game picks one of the nodes in the graph. The chosen node picks a "choice" (also Yes or No) and the agent is committed to that "choice" for the rest of the game. The game ends when all nodes have picked a "choice."

We keep the same scheduler behavior as in the previous paper. The schedulers utility is based off of the number of nodes that choose Yes at the end of the game. We assume the scheduler can change its plan after it sees what a node chooses. We also assume it has full knowledge of the nodes strategies, as well as knowledge of the full graph, and any game parameters. The only aspect the scheduler does not know is the individual nodes types.

The nodes utility is the sum of all of their neighbors that match their choice plus some constant (π) if their choice matches their type. Strategic nodes make decisions to maximize their expected profit. We assume strategic nodes know the schedulers decision making (or utility function), the graph structure, and the probability that a node is assigned Yes type (p), but not any other nodes type. To keep the model interesting, we restrict $p < 0.5$. Similarly myopic nodes only try to maximize their current utility, ignoring any nodes that has not made a decision.

We consider games where either all nodes in the graph are strategic or myopic, and want to see what graphs / game parameters contribute to significant differences between the expected number of Yesses when nodes are strategic versus myopic.

We formally specify the model as follows

$G(V, E)$	social network
$\lvert V\rvert = n$	
$c_i \in \{Y, N\}$	choice of node i
$t_i \in \{Y, N\}$	type of node i
$p(t_i = Y) = p < 0.5$	
$u_i(\mathbf{c}) = \pi \mathbb{I}[t_i = c_i] + \sum_{j \in N(i)} \mathbb{I}[c_j = c_i]$	node utility
$sched(\mathbf{c})$	schedule function

where $\mathbb{I}$ is the indicator function, and N is the neighbor function. The schedule function returns the next node to pick given the choices that have been made so far.

The full game can be described by the following algorithm. The scheduler picks an optimal first node in the graph. That node chooses either Yes or No based on all of the parameters and its knowledge. Based on this result the scheduler picks a new node, and so on until every node has been picked and the game finishes.

Given this game, we are interested in quantifying

$$\frac{\mathbb{E}[\#Y | strategic]}{\mathbb{E}[\#Y | myopic]}$$

for a given family of graphs with different game parameters.

Is seems likely that computing the strategic choices and payouts is a hard problem, but we will show an algorithm that can compute an optimal online schedule and calculate $\mathbb{E}[\#Y | stragetic]$ in polynomial time for arbitrary block models.

A graph is a k-block model if the vertices can be divided into k disjoint subsets such that every node in the same subset has incoming and outgoing edges to the same set of nodes. Another way to view this property is by looking at the adjacency matrix of the graph. A block model graph can be written as a symmetric block matrix with k^2 blocks. For the sake of the algorithm, we will define

$$\{s_i\}_{i=1..k}$$

as the number of nodes in each block.

An example of a 2-block model would be the star graph, which can be expressed in adjacency matrix form as

$$\begin{array}{r} 1\{ \\ n-1\{ \end{array} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

and has $\mathbf{s} = \{1, n-1\}$.

3 Algorithm

Our algorithm uses two lookup tables (one for the scheduler, and one for the nodes) to determine the optimal play given the relevant statistics of the game at every decision time. Each table has a dimension for every relevant play statistic for the deciding agent, and contains their optimal choice as well as the expected number of Yeses in each block after that choice is made (which is necessary for callers to decide on their optimal decision, as expected payoffs are only a function of the expected number of Yeses at the end of the game).

The two tables recursively reference each other to calculate their entries, where each entry is the choice, and the expected number of Yesses at the end of the game in each block conditioned on that choice. If an entry is already calculated, then the lookup is constant time, otherwise the lookup may cause a cascade of calculations, which is bounded by the polynomial time to compute both tables.[1]

We will represent the tables as recursive functions, with the implicit understanding that we're caching results to get a polynomial time algorithm.

To determine the optimal block choice for the scheduler, the decision function only needs to know, ho many nodes from each block have already gone, and what proportion (or number) of the nodes have chosen Yes. Because nodes within a block are exchangeable, this information completely characterizes the state of the game at the time of scheduler choice.

Similarly, to determine the optimal choice (Yes or No) of a node, the node only needs (has available) its own type, what block it is in, the number of nodes in each block that have already chosen, and the number of those nodes that have chosen Yes.

We formally define the optimal scheduler choice function as

$$\text{sched}(\{c_i\}, \{y_i\})$$

and the optimal node choice function as

$$\text{node}(t, b, \{c_i\}, \{y_i\})$$

where

t	Node's type (Yes or No)
n	Node block
$\{c_i\}$	Number of already chosen nodes by block
$\{y_i\}$	Number of Yes chosen nodes by block

The scheduler needs to check the possible outcome given it picks a Yes or a No type node from each of the k blocks that have a most one node possible to

[1]It is possible that instead of looking these up on the fly, he table could be precomputed to speed up the complexity slightly, but we believe that this speedup would be very slight, and would make the algorithm much more complicated

pick. It then marginalizes out the type of the node to calculate the expected number of Yesses for each choice of block. The optimal choice is simply the one that maximizes the expected number of Yesses.

A node needs to check its own expected final utility given it chooses Yes or No, and choose the choice that maximizes it. The node can query the scheduler function to determine the expected number of Yesses in each block conditioned on the node choosing Yes or No. The node then trivially calculates its expected utility for each choice, and picks the maximum.

Completely calculating the results of these functions is trivially simple, and in doing so completely finds the optimal online schedule and optimal node choice along that schedule.

4 Complexity

See Section 3 for an explanation of the algorithm.

Lookups are constant time, so the total time complexity is going to be bounded by the number of entries in the table times the number of table lookups it takes to compute one entry in the table.

The size of the node table is trivial to upper bound. There are $2k + 2$ dimensions: the node's type (2 values), the node's block (k values), k dimensions for the number of nodes in each block that have gone before ($\leq s_i + 1 | i = 1..k$), and k dimensions for the number of nodes in each block that have gone before and selected yes ($\leq s_i + 1 | i = 1..k$). This leads to the following statement,

$$\begin{aligned} NodeTableSize &= o\left(2k \prod_{i=1}^{k} s_i + 1 \prod_{i=1}^{k} s_i + 1\right) \\ &= o\left(k \prod_{i=1}^{k} s_i^2\right) \end{aligned}$$

If we further assume that $\ell \leq k$ of the blocks are size order n and the rest are constant size, then the size of the table reduces to

$$NodeTableSize = o\left(kn^{2\ell}\right)$$

Each node table entry computation takes 2 calls to the sched function, and then does computation of the expected number of Yeses which is order k, making the total complexity for the node table given the sched table

$$\begin{aligned} NodeTableComplexity &= o\left(k^2 \prod_{i=1}^{k} s_i^2\right) \\ &= o\left(k^2 n^{2\ell}\right) \end{aligned}$$

The size of the scheduler table is also trivial to upper bound. There are $2k$ dimensions similar to the node table. By similar reasoning

$$\begin{aligned} SchedTableSize &= o\left(\prod_{i=1}^{k} s_i + 1 \prod_{i=1}^{k} s_i + 1\right) \\ &= o\left(\prod_{i=1}^{k} s_i^2\right) \end{aligned}$$

If we make the same assumption about block sizes this reduces to

$$SchedTableSize = o\left(n^{2\ell}\right)$$

The complexity for each entry in the scheduler table is linear in k, because it makes $2k$ lookups to the node table. Therefore the total complexity for constructing the scheduler table is bounded by

$$\begin{aligned} SchedTableComplexity &= o\left(k \prod_{i=1}^{k} s_i^2\right) \\ &= o\left(kn^{2\ell}\right) \end{aligned}$$

This leaves us with final time complexity of

$$\begin{aligned} RunningTime &= o\left(k^2 \prod_{i=1}^{k} s_i^2\right) \\ &= o\left(k^2 n^{2\ell}\right) \end{aligned}$$

Similarly the node table is the larger of the two tables, and has space complexity equal to time complexity.

5 Results & Discussion

We implemented our dynamic program algorithm and ran it on several categories of graphs in order to guide intuition for theoretical work regarding this problem. Our primary concern was the difference in expected number of Yes decisions for the optimal adaptive schedule given strategic or myopic users. We aim to characterize situations under which strategic users give more Yes decisions than myopic users, or vice versa. Our secondary concern was testing the predicted computational complexity of our algorithms.

We test a star graph, a clique, and a special graph we call a cloud graph. A cloud graph consists of two singular "outer" vertices of degree a and b respectively, one singular "inner" vertex of degree $a + b$, and two clouds of size a and b respectively, with each vertex of degree two. Each of the "outer" vertices is connected to every node in a distinct cloud. The singular "inner" vertex is connected to every node in both clouds. This graph, under certain parameters, provably obtains more Y adoptions for strategic users.

We examine the ratio of expected number of Ys between strategic and myopic while varying the parameters p, π, n. Colorplots of the ratio for the set value $n = 10$ but varying p, π can be found in Figure 1. The primary lesson we learned from Figure 1 is that the behavior on these graphs is extremely complex. There appears to be no clear general characterization of how the ratio will behave.

Though these plots provide little obvious insight into the general case of graphs, we can extract valuable information about specific behaviors and provide some upper and lower bounds on the extremes of the ratio.

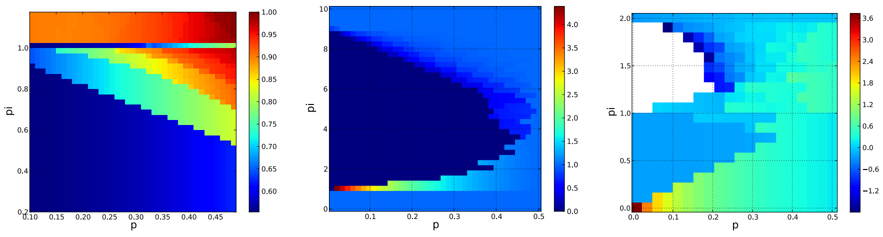
The ratio is increasing in p for the star graph, decreasing in p for the clique, and both (depending on π) for the cloud graph. Even more strangely, the ratio is (mostly) increasing in π for the star graph but depends on the p value for the clique and cloud graphs. We see markedly different behavior when $\pi = 1$. This is the point at which agents are indifferent between choosing their own type and going with the expected majority. This indifference is unique to $\pi = 1$, as our tie breaking methods lead to significant differences in outcome (nodes pick No in the event of a tie).

We learn several things regarding the question of whether strategic agents can ever give more Y adoptions than myopic agents. Data for the star graph supports our theoretical result that myopic is always better on the star graph (and thus the ratio is bounded by 1). The clique plot suggests an interesting result. For small p, π, strategic is better than myopic. This is an unexpected, but explainable result. For $\pi = 1 + \epsilon$, only 1 Y decision is required to start a Y cascade with strategic users, but 2 Y decisions are required with myopic users. So the ratio can be as large as $\frac{1}{p}$. We also confirm some cascade proofs we've had for the clique and cloud graphs, namely that we can achieve either a Y or an N cascade on the cloud graph, depending on the parameters, and that for many parameters the clique achieves an N cascade.

We leave out plots which vary n, in the interest of space, but these plots give similarly varied results.

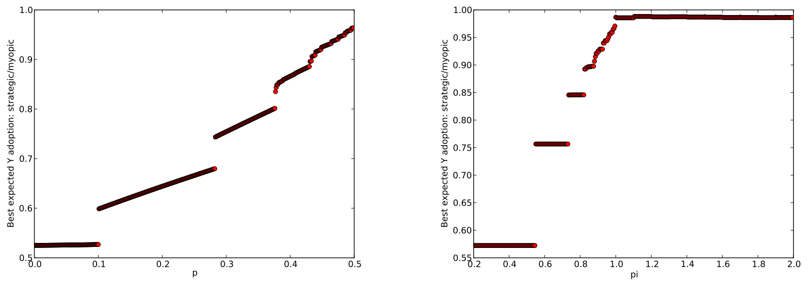
Careful analysis of plots in Figure 1 reveals that our graphs have sharp bands of color in which small changes in p or π result in drastic changes in ratio. We graph numerical values in Figure 2 in order to investigate this phenomenon on the star graph. Drastic ratio changes correspond to changes in fundamental behavior of strategic nodes on the star. For example, as p or pi get larger, the situations under which a strategic Y type edge node might choose N change. For low enough π (on the star graph), Y type nodes will choose N after only one edge node has chosen N. As π rises, this stubbornness threshold also rises in discrete jumps as seen on the graph.

Lastly, we investigate the computational complexity of our algorithm in Figure 3 by measuring computation time over varied graph sizes. We time the star graph, which has one block that scales with n, and the cloud graph, which has two blocks that scale with n. As predicted, running time is quadratic for the star graph and quartic for the cloud graph.



Star Clique Cloud[2]

Figure 1: Ratio of strategic:myopic Y adoptions for varying p, π.



Varying p Varying pi

Figure 2: Ratio of strategic:myopic Y adoptions for varying p, π on star graph.

6 Future Work

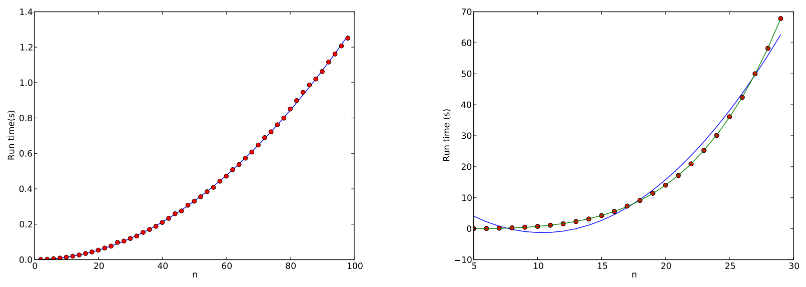
Our results are interesting and certainly show that the difference between optimal play with myopic and strategic users is a complex space even very simple block graphs. However, there are still a number of interesting unexplored questions that we have not looked at in the scope of the problem.

Despite the fact that obvious patterns don't exist in the performance ratio, it is still possible that the performance ratio can be bounded in certain circumstance beyond the star graph. Proving bounds on the performance ratio could help characterize the discontinuous regions in the plots where myopic superiority switches in favor of strategic agents.

We have only shown a polynomial time algorithm for graphs where the number of blocks is constant with respect to N. A further and important step would be to come up with an algorithm for general graphs, or to reduce the problem to another complexity class like PPAD or NP.

The algorithm as it is presented only works for online scheduling. This is a necessary condition to allow the dynamic programming to work, however ongo-

[2]Log scale. White $= -\infty$ or $\log 0$



Star Cloud

Figure 3: Code runtime for varying graph size. Blue line is best quadratic fit, green line is best quartic fit.

ing research may suggest that for offline scheduling, myopic agents will always permit a higher number of Yes nodes. Looking at performance bounds under offline scheduling could pose an viable avenue to study performance difference where online schedules became very complicated. It is not apparent how to compute an optimal offline schedule efficiently for a block graph or otherwise.

References

[1] W. Brian Arthur. Competing technologies, increasing returns, and lock-in by historical events. *The Economic Journal*, 99(394):pp. 116–131, 1989.

[2] Flavio Chierichetti, Jon Kleinberg, and Alessandro Panconesi. How to schedule a cascade in an arbitrary graph. *EC*, pages 355–368, 2012.

[3] Michael Kearns, Michael L. Littman, and Satinder Singh. Graphical models for game theory. pages 253–260, 2001.

[4] Stephen Morris. Contagion. *The Review of Economic Studies*, 67(1):57–78, 2000.

[5] Tom A.B. Snijders and Krzysztof Nowicki. Estimation and prediction for stochastic blockmodels for graphs with latent block structure. *Journal of Classification*, 14:75–100, 1997. 10.1007/s003579900004.

[6] Yuchung J. Wang and George Y. Wong. Stochastic blockmodels for directed graphs. *Journal of the American Statistical Association*, 82(397):8–19, 1987.

[7] Duncan J. Watts. A simple model of global cascades on random networks. *Proceedings of the National Academy of Sciences*, 99(9):5766–5771, 2002.