2011

Influence maximization on families of graphs

Andrei Mouravski

Influence Maximization on Families of Graphs

by

Andrei Mouravski

A Thesis Submitted in Partial Fulfillment of the Requirements for the
Degree of Master of Science
in Computer Science

Supervised by

Professor Christopher M. Homan, Ph.D.
Department of Computer Science
B. Thomas Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, New York
February 24, 2011

Approved by:

Christopher M. Homan, Ph.D., Professor
Thesis Advisor, Department of Computer Science

Ivona Bezáková, Ph.D., Professor
Reader, Department of Computer Science

Stanisław P. Radziszowski, Ph.D., Professor
Observer, Department of Computer Science

Thesis Release Permission Form

Rochester Institute of Technology
B. Thomas Golisano College of Computing and Information Sciences

Title:

Influence Maximization on Families of Graphs

I, Andrei Mouravski, hereby grant permission to the Wallace Memorial Library to reproduce my thesis in whole or part.

Andrei Mouravski

Date

Dedication

To Ryan, who showed me the importance of every moment.

Acknowledgments

I am grateful for my committee, the Computer Science Department, my family, many Bothans who died to bring us this information, the Dr. Who crew, for wibbly-wobbly, timey-wimey . . . stuff,

and Laurel, for always supporting me.

Abstract

Influence Maximization on Families of Graphs

Andrei Mouravski

Supervising Professor: Christopher M. Homan, Ph.D.

We examine computing the maximum expected influence on paths and trees using the independent cascade model. We designed a polynomial time method for determining the expected influence from any initial state on the independent cascade model on acyclic influence graphs in $\mathrm{O}\Big(|V(G)|^2 \cdot \max\{|V(G)|, |V(E)|\}\Big)$ time. We designed a polynomial time program that would computes the maximum expected influence and optimal initially selected nodes on arbitrary paths in $\mathrm{O}\Big(\frac{|V(T)|}{\varepsilon}^6\Big)$ on approximating the maximum expected influence on arbitrary trees with absolute error of at most $\big(\varepsilon \cdot 2|V(T)|\big)$.

Contents

Chapter 1

Introduction

Diffusion describes the process by which an idea, rumor, or infection can spread through society [3, 10, 15, 18]. In the social sciences, we can trace the progression of an idea or an innovation between different groups, such as civilizations or businesses. In epidemiology, we can map how diseases spread from individual to individual. In marketing, it is useful to know how to best target our advertising dollars to garner the most impressions. This last example is the basis of this thesis.

Domingos and Richardson provided a description of the influence maximization problem as it applies to data mining. [3]. Instead of focusing on the specifics of markets or networks, this work developed the idea of an individual's "network value" or influence, which is the net value a marketer gets in further marketing by simply seeding the individual with a product. Kempe, Kleinberg, and Tardos introduced the function σ, which is the expected number of reached individuals after selecting the set of initial individuals [15]. The goal of our research is maximizing this function.

Our research addresses the computational complexity of the problem that asks, given a model of diffusion with underlying graph in a family of graphs, what is the maximum expected number of nodes reached by seeding the diffusion with a fixed initial cost [14]. This thesis explores whether there is a polynomial time algorithm for maximizing the extent of diffusion by optimally choosing the initial nodes. It is known to be an NP-hard problem to determine the maximum expected diffusion given any set of nodes in the general case [4, 14]. It is unknown, however, whether a polynomial time algorithm can determine the maximum expected diffusion on a family of graphs or even whether we can approximate the optimum in polynomial

time.

We explore the behavior of diffusion models on several families of graphs, where by "graph family" we mean a set of graphs that all share a specific property. Many authors have studied diffusion, but most research has been based only random graphs, and not families of graphs [3, 4, 5, 8, 14]. There is a general framework that includes many diffusion models [15, 16], so for clarity our research examines one particular model on various families of graphs. When restricted to a specific family of graphs, other NP-Hard problems are solvable in polynomial time [1], so we expect solving the influence maximization problem on families of graphs will yield comparable results.

Our research either provides a polynomial time algorithm for determining the maximum expected diffusion on a family of graphs given the initial cost or describes a polynomial time approximation scheme for the optimum value.

Our research may have some ties to the work of Douglas Heckathorn. Heckathorn proposed a method of sampling called respondent-driven sampling [13], which we will define in Section 2.2. This method of sampling is highly effective on populations that are not otherwise easily reached. Furthermore, this method is designed to overcome social biases prevalent in other forms of sampling. RDS attempts to generate a sample that exhibits the same demographical properties as the network itself [7, 11]. We will briefly explore respondent-driven sampling as it pertains to diffusion and influence maximization.

In Chapter 2 we will define diffusion and many models that simulate diffusion. This information provides useful background into the process and properties of diffusion, which we will use in constructing maximization schemes. Section 2.4 details the decision problem related to our work and the optimization problem that we will be delving into.

In Chapter 3 we will provide the results of our work including a polynomial time algorithm for determining the expected influence given a set of initially selected nodes on acyclic influence graphs, which we will define within. We also detail a polynomial time algorithm for determining the maximum expected influence on a path. Lastly, we have constructed

a polynomial time approximation scheme for estimating the maximum expected influence on trees. An associated exponential time solution for the optimization problem on trees is also included.

Chapter 4 contains our conclusions and explains potential further research into this subject.

Chapter 2

Background and Related Work

In this chapter we will describe the foundation of our research. In Section 2.1, we will look at diffusion models and the process of diffusion. We will cover a variety of models that have been researched and will illustrate the differences between various models. As diffusion is the core of our research we provide as much background as possible. Our results utilize just one model of diffusion, the independent cascade model, but others are defined so as to contrast various models of diffusion so that the reader may better understand related research.

In Section 2.2, we outline Respondent-Driven Sampling and its ties to our research. Though we do not directly link this topic to our results, we provide it as further context. We also outline several relevant graph theory terms that base our understanding of the families of graphs on which we will be applying our algorithms.

Section 2.3 provides a brief summary of graph theory terms necessary for understanding our results.

We provide a formal definition for both the influence maximization optimization problem and the influence maximization decision problem in Section 2.4.

In the final Section, 2.5, we outline the problem of approximating the maximum expected influence and related work. As we will be proving our own approximation result later in this paper, Section 2.5 is provided as a baseline.

2.1 Diffusion Models

Diffusion is the process by which information passes from neighbor to neighbor. Real world examples include viral marketing, innovation of technologies, and infection propagation. Diffusion models are the framework on which diffusion occurs.

Definition 2.1.1. *[16] A **diffusion model** is a graph $G = \{V, E\}$ along with a collection of activation functions $\mathcal{F} = (f_v)_{v \in V}$, where f_v is a $\{\emptyset, \{v\}\}$-valued function on 2^V.*

The output of a function f_v is a random variable based on the activation function.

Vertices on this graph are usually individuals and the activation function models the influence individuals exert on others. The activation function usually depends only on the neighbors of v, denoted $N(v)$. This means that $f_v(S) = f_v(N(v) \cap S)$ [16].

Definition 2.1.2. *[16] **Diffusion** is the process on a diffusion model M, **S** = $(S_t)_{t=0}^{n-1}$ started at $S \subseteq V$:*

1. *set $S_0 = S$*
2. *for $t > 1$ set $S_t = f(S_{t-1}) =_{\text{def}} \bigcup_{v \in V} f_v(S_{t-1})$*

The set of nodes activated at the end of diffusion is denoted as $\varphi(S) = \bigcup_{t=0}^{n} S_t$.

Diffusion occurs in time steps t. At each time step, all previously activated nodes remain activated and individuals are either activated or deactivated based on the activation functions. Diffusion can run on a fixed number of time steps or indefinitely. Diffusion is said to have stopped when the set of activated nodes in time step t_k is the same as the set in time step t_{k+n} for all $n \geq 1$.

2.1.1 Threshold Models

One class of diffusion models adds an influence threshold to each individual, which, when overcome, triggers the individual to be activated. The idea behind these models was first proposed by Grannovetter [10] and Schelling [18] mainly in the context of the social sciences. There is a cumulative effect of these models, as it takes a critical number of influential neighbors to activate an individual.

General Threshold Model

This model was defined by Kempe, Kleinberg, and Tardos [15] and Mossel and Roch [16].

Definition 2.1.3. *[15, 16] The **general threshold model** is a diffusion model with*

- *a set of threshold values $(\theta_v)_{v \in V}$, where θ_v is in the range $[0, 1]$*
- *node v being activated if $f_v(S) \geq \theta_v$, where S is the set of neighbors of v.*

The activation function on the general threshold model depends on the activated neighbors of v. There is an assumption of monotonicity on this model made to reflect that adding active neighbors to a node increases likelihood of the node being activated.

Definition 2.1.4. *[16] A function $f : 2^V \to \mathbb{R}$ is **monotone** if $f(S) \leq f(T)$ for all $S \subseteq T \subseteq V$.*

This property captures that activating more nodes will always have an increasing effect on the nodes that will be activated at a future time.

Linear Threshold Model

The linear threshold model is a specialized form of the general threshold model. The linear threshold model is more often used in marketing research [3, 10, 18]. This model imposes the property that the sum of weights to a node is bounded by 1.

Definition 2.1.5. *The **linear threshold model** is a diffusion model with all of the properties of the general threshold model with*

- *a set of weights* $(b_{v,w})_{v,w \in V}$ *with the property* $\sum_{w \in N(v)} b_{v,w} \leq 1$
- *activation function of the form* $f_v(S) = \sum_{u \in S,\, a_u = 1} b_{v,u}$*, with* $f(\emptyset) = 0$.

Note that this model is deterministic unlike the general model. We know whether a node is active or not by just counting the sum of the weights of all active neighbors.

History-Sensitive Cascade Model

The history-sensitive cascade model, designed by Foster and Potter [5], is essentially a reformat of the linear threshold model and is not a different diffusion model itself. In their research into the spread of influence, Foster and Potter propose the idea that the probability of a node being activated increases the longer the node is in contact with other activated nodes. Since at every time step more neighbors can be added, while the combined influence never goes down, the probability that any node is activated increases with each new neighbor added. This reflects the monotonic property of the linear threshold model.

Foster and Potter studied the exact effects of diffusion over time on the probability that any node would be activated at time step k. They studied this effect on tree-structure graphs and also on general graphs and proposed algorithms for determining these probabilities. To attain the probability of a node being activated at any given time step, a Markov chain model is used.

Definition 2.1.6. *[5] A **Markov chain** is a sequence of random variables* $X_1, X_2, X_3, \ldots$ *with the property that* $Pr(X_{n+1} = x | X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) = Pr(X_{n+1} = x | X_n = x_n)$.

A Markov chain is a collection of states with transitions between states such that the probability of transitioning to any state from any other state depends only on the current state. Foster and Potter use a Markov chain model that encode sets of active nodes in binary strings and then create a

transition matrix that maps the probability of transitioning from any set of activated nodes to any other set. By iterating over this transition matrix, it is possible to find the exact probability of any node being activated at any time step for any arbitrary graph [5].

2.1.2 Cascade Models

Cascade models of diffusion give each individual the ability to influence their neighbors as soon as they are activated. This is opposed to the threshold models that rely on a cumulative effect.

General Cascade Model

This model was designed by Kempe, Kleinberg, and Tardos [15] as a general form of the cascade model. This model has the property that the more nodes that have attempted to influence a node, the less likely the node is to be activated.

Definition 2.1.7. *[15] The **general cascade model** is a diffusion model with the following properties:*

- *nodes are **live** at time t if they were activated in time $t-1$*
- *a collection of probability functions $\mathcal{P} = (p_v)_{v \in V}$, where p_v is a $[0,1]$-valued function on 2^V*
- *activation function of the form*

$$f_v(W) = \begin{cases} 1, & \text{with probability } p_v(W) \\ 0, & \text{otherwise} \end{cases}$$

  where $W \subseteq S$ and every $w \in W$ is live at time t

- *node v being activated in time t if $f_v(W) = 1$, where W is the set of neighbors of v live at time t*
- *the order-independence property, defined below.*

Note that each of the following definitions use p_v as an element of $\mathcal{P}$ and are defined over all $v \in V$. Likewise for f_v as an element of $\mathcal{F}$ defined over all $v \in V$.

Definition 2.1.8. *[15] The **order-independence property** states that when $\sigma : \{1, \ldots, r\} \to \{1, \ldots, r\}$ is a permutation function and $\{u_1, \ldots, u_r\}$ and $\{u_{\sigma(1)}, \ldots, u_{\sigma(r)}\}$ are two permutations of T, and $T_i = \{u_1, \ldots, u_{i-1}\}$ and $T_i' = \{u_{\sigma(1)}, \ldots, u_{\sigma(i-1)}\}$, then*

$$\prod_{i=1}^{r}(1 - p_v(\{u_i\} \cup S \cup T_i) = \prod_{i=1}^{r}(1 - p_v(\{u_{\sigma(i)}\} \cup S \cup T_i')$$

for all sets S disjoint from T.

The probability of a node u influencing a node v depends on the set S of nodes that has already attempted to influence v. However, the order-independence property states that the probability of u activating v does not depend on the order of nodes in the set S that have previously attempted to activate v.

General Cascade and General Threshold Equivalence

The general cascade model has been shown to be equivalent to the general threshold model [15, 16] under the following mapping:

- for the probability function in the general cascade model:

$$p_v(\{u\} \cup S) = \frac{f_v(S \cup \{u\}) - f_v(S)}{1 - f_v(S)},$$

- for the activation function in the general threshold model:

$$f_v(S) = 1 - \prod_{i=1}^{r}(1 - p_v(\{u_i\} \cup S_{i-1}),$$

  where $S = \{u_1, \ldots, u_r\}$ and $S_i = \{u_1, \ldots, u_i\}$.

This effectively says that by choosing the edge weights in either model, an instance of the general threshold model may be transformed into an instance of the general cascade model. This mapping ties the two models together and shows that diffusion is an equally hard problem on either model. Therefore, conclusions on one model also apply to the other model.

Decreasing Cascade Model

The decreasing cascade model was also defined by Kempe, Kleinberg and Tardos [15] and is an extension of the general cascade model with the property that the more nodes that have attempted to activate a node, the less probability there is that the node becomes activated.

Definition 2.1.9. *[15] The **decreasing cascade model** is a diffusion model with all of the properties of the general cascade model with the additional property where $p_v(\{u\} \cup S) \geq p_v(\{u\} \cup T)$ whenever $S \subseteq T$.*

Independent Cascade Model

This model was initially investigated by Goldenberg, Libai, and Muller in the context of marketing [8, 9] and was defined by Kempe et al. [14]. Along with the linear threshold model, this model is classically used for studying diffusion on networks. It exists as a special case of the decreasing cascade model.

Definition 2.1.10. *[14] The **independent cascade model** is a diffusion model with all of the properties of the decreasing cascade model with the additional property that the $p_v(\{u\} \cup S) = p_v(\{u\})$ for all sets $S \subseteq V$.*

This means that the probability of a node u influencing a node v is independent of the set of nodes S that has attempted to influence v.

Since we will be using this model for the remainder of our research, it is helpful to define some shorthand. We can look at this model as a set of edge probabilities on a graph.

Definition 2.1.11. *On the independent cascade model, an **edge probability**, $b_{u,v}$ is the probability that a node u has to infect v whenever u is infected.*

Note that $b_{u,v}$ does not necessarily equal $b_{v,u}$ and in fact, it will be the case in certain situations in our research that if $b_{u,v}$ is non-zero, that $b_{v,u}$ is 0.

It should be noted that the independent cascade model has the property that a node has exactly one time step in which it is infected to infect other nodes. That is, each node is infectious for exactly one time step and then can no longer be infected, nor can it infect any other nodes.

2.2 Respondent-Driven Sampling

We now provide a background of Respondent-Driven Sampling. While not directly related to our research, Respondent-Driven Sampling provided much of the inspiration for this thesis.

Respondent-Driven Sampling is a form of sampling espoused most notably by Douglas Heckathorn [13, 11, 12], but also by others [6, 19, 22, 21]. Sampling in this case refers to the method of attaining a representative subset of a population of people. The goal of sampling in general is to choose a subset such that it displays as many properties of the superset as possible. The problem with sampling in most cases is that the subset derived is not representative of the population as a whole. There are many biases evident in sampling, such as choosing too frequently those subjects that want to be sampled, choosing subjects too selectively, or choosing subjects too broadly [13]. Respondent-Driven Sampling, or RDS, is concerned with a problem even greater in that it targets "hidden populations," that is populations where no public reference frame or directory exists. Some types of hidden populations are drug users, men who have sex with men, and members of criminal organizations. Such populations require a different type of sampling to account for the reluctance of individuals in joining the study.

There have been various methods for correcting for the biases in these samples, such as snowball sampling, key informant sampling, and chain referral. Each of these sampling methods has inherent issues and introduces biases [13]. The main issues to be addressed are threefold: 1) the samples acquired are dependent on the initial population, 2) only more cooperative subjects are in the sample, which is not indicative of the population as a

whole, and 3) privacy concerns of subjects may skew the sample.

Respondent-driven sampling utilizes a Markov chain process to create a sample of the population that is representative of the population as a whole [7, 13]. The exact method for using Markov chains is outside the scope of this paper and is outlined in [7]. The key conclusion of RDS research is that respondent-driven sampling is able to choose a good sample. For this reason, RDS has been used in numerous contemporary studies [6, 7]. While there are reasons to believe that there are problems inherent in respondent-driven sampling [6, 11], there is a possible relationship to diffusion. While this relationship is outside the scope of this thesis, RDS and diffusion are both involved with reaching as many individuals as possible, and so further research into this subject is necessary.

2.3 Graphs

Graph theory is the language of networks and it forms the backbone for any study of diffusion.

Definition 2.3.1. *[23] A **graph**, G, is defined as a triple of a vertex set, $V(G)$, and edge set, $E(G)$, and a relation between them that connects each edge to two vertices.*

A graph is the basic representation of a collection of objects that are related in pairs. The vertices in a graph, also called nodes, can represent individuals, groups, computers, or any kind of object or idea. Edges depict connections between vertices and may be any form of association or relation. The neighbors of a vertex refer to all vertices that are connected by an edge to the vertex in question. The set of neighbors of a vertex is called the neighborhood.

A family of graphs is a set of graphs that all share a property. In this paper we will be specifically discussing the following families of graphs, which are provided here as a reference, even if results cannot be attained yet:

- Paths

- Trees
- Bipartite Graphs
- Planar Graphs
- Graphs of Bounded Maximum Clique

Definitions for many basic graph theory terms and concepts can be found in any introductory graphy theory textbook such as [23]. We will not reprint these definitions here.

2.4 Influence Maximization of Diffusion

To better understand the underlying ideas behind diffusion and social networks, we study the problem of influence maximization. Influence is the effect of a set of activated nodes on other nodes. Influence denotes the expected number of nodes activated after the diffusion process.

Definition 2.4.1. *[16] Given a graph $G = (V, E)$, a set $A \subseteq V$, and a diffusion model, **influence**, or $\sigma(A)$, with respect to the graph and diffusion model, can be expressed as*

$$\sigma(A) = \mathbb{E}[|\varphi(A)|]$$

The expectation maximization problem is alternatively called the spread maximization problem or influence maximization problem. Each of these refers to maximizing influence. We have a separate decision problem based on the diffusion model parameters, that is what model of diffusion that is used.

Definition 2.4.2. *The **influence maximization problem** is a problem with diffusion model parameters M that asks, given a graph $G = (V, E)$ and an integer $c \leq |V|$, what is $max(\sigma(A))$ where $A \subseteq V$ and $|A| \leq c$?*

Definition 2.4.3. *The **influence maximization problem decision problem** is a problem with diffusion model parameters M that asks, given an instance of the influence maximization problem and an integer k, is $max(\sigma(A)) \geq k$?*

This problem is NP-hard , and so attempts have been made at approximating the value of $\sigma(A)$ [15].

2.5 Approximation

For a diffusion model with a submodular activation function and a submodular $\sigma(\cdot)$ function, a greedy hill-climbing algorithm approximates the optimum within a factor of $(1 - 1/e - \epsilon)$ for any real number ϵ, as shown by Kempe, Kleinberg, and Tardos [14] and Mossel and Roch [16] after the results of Nemhauser, Wolsey, and Fisher [2, 17]. By "greedy hill-climbing algorithm" we mean an algorithm where, at every step, we add to the output set the element that currently has the highest value.

Submodularity is a property on a diffusion model that states that the influence gained from adding nodes to the infected set decreases or stays the same as the set becomes larger. This condition can be read as a principle of "diminishing returns," where the value of adding a node to the infected set decreases based on the size of the infected set. It is equivalent to say the activation function of the model is submodular.

Definition 2.5.1. *[15, 16] A function $f : 2^V \to \mathbb{R}$ is **submodular** if for all $S, T \subseteq V$,*

$$f(S) + f(T) \geq f(S \cap T) + f(S \cup T).$$

Equivalently, if $S \subseteq T$ and $v \in V$, then

$$f(T \cup \{v\}) - f(T) \leq f(S \cup \{v\}) - f(S)$$

Definition 2.5.2. *[20] A submodular function f is **normalized** if $f(\emptyset) = 0$.*

The normalized submodular threshold model exemplifies these properties and was designed by Kempe, Kleinberg, and Tardos [15] with an elaboration by Mossel and Roch [16]. Note that the definition of normalized for submodular functions differs from the definition for diffusion models. Below, the normalized property has the same function, but is defined differently.

Definition 2.5.3. *[15, 16] The **normalized submodular threshold model** is a diffusion model with all of the properties of the general threshold model with the added property that each activation function f_v satisfies the normalized submodular property:*

$$\frac{f_v(S \cup \{i\}) - f_v(S)}{1 - f_v(S)} \geq \frac{f_v(T \cup \{i\}) - f_v(T)}{1 - f_v(T)}$$

for all $S \subseteq T$.

The submodular property is significant because it imposes an ordering on different sets of infected nodes. This property of the activation function shows that adding nodes to the infected set cannot decrease the expected influence. This leads to a greedy-hill climbing algorithm proposed by Nemhauser, Wolsey, and Fisher [2, 17] that approximates the optimum value for activation function within a factor of $(1 - 1/e)$.

Theorem 2.5.4. *[2, 17]*

For a non-negative, monotone submodular function f, let S be a set of size k obtained by selecting elements one at a time, each time choosing an element that provides the largest marginal increase in the function value. Let S^* be a set that maximizes the value of f over all k-element sets. Then $f(S) \geq (1 - 1/e)f(S^*)$; in other words, S provides a $(1 - 1/e)$-approximation.

This theorem is used by Kempe, Kleinberg, and Tardos [14] to create a bounded approximation for the optimum value of the influence maximization problem. Nemhauser, Wolsey, and Fisher assume that the greedy algorithm can evaluate the underlying function exactly, but it is unknown whether it is possible to determine $\sigma(\cdot)$ exactly in polynomial time [14]. However, Kempe et al. generate arbitrarily close approximations of $\sigma(A)$ by simulating diffusion and then sampling the resulting sets [14]. This leads to an approximation result:

Theorem 2.5.5. *[15, Theorem 1] Let A^* be the the set maximizing $\sigma(\cdot)$ among all sets of k nodes.*

1. *If the optimal v_i is chosen in each iteration, then the greedy algorithm is a $(1 - 1/e)$-approximation, i.e., the set A found by the algorithm satisfies $\sigma(A) \geq (1 - 1/e) \cdot \sigma(A^*)$.*

2. *If the node v_i is a $1-\epsilon$ approximate best node in each iteration, then the greedy algorithm is a $(1 - 1/e - \epsilon')$-approximation, where ϵ' depends on ϵ polynomially.*

Kempe et al. prove this result by showing, on the models used, the decreasing cascade model and the normalized submodular threshold model, that $\sigma(\cdot)$ is submodular. This then guarantees that the results of Nemhauser et al. can be applied both to the $\sigma(\cdot)$ function and diffusion as a whole.

Kempe, Kleinberg, and Tardos executed this greedy algorithm on real data and show on general graphs that the approximation matches the theoretical value [14].

Chapter 3

Results

Our goal is to find a polynomial time algorithm for determining the maximum expected influence on graphs. It is NP-hard to determine the maximum expected influence on all graphs, but, using the independent cascade model, we have polynomial time results on all acyclic influence graphs and paths and we have a polynomial time approximation scheme for all trees.

In this chapter, we will first define several terms that are used in our algorithms. In Section 3.1 we provide preliminary results on all graphs using the independent cascade model. These results are built upon in Section 3.2, where we provide a polynomial time algorithm for determining the expected influence given an initial set on all graphs using an acyclic influence graph. We define this new graph property on an instance of a diffusion model in Section 3.2.

In Section 3.3 we produce a polynomial time algorithm for determining the maximum expected influence using the independent cascade model on any arbitrary path.

Lastly, in Section 3.4 we provide the main result of this thesis by showing, using the independent cascade model on any arbitrary tree, that we are able to provide a polynomial time approximation for the optimum value to the influence maximization problem to any arbitrary constant. We first prove this result for binary trees and then generalize our program to arbitrary trees.

3.1 Preliminary Results on the Independent Cascade Model

We will first provide a partial result using the independent cascade model on arbitrary graphs. We utilize the independent cascade model's independence

property to show that there is a formulation for attaining the probability that any node is in its infected state, given that it was uninfected in the previous time step. This formulation is provided as a base for the stronger result in Section 3.2.

3.1.1 Definitions

A few definitions are necessary to reach this actual result. State property information is implicit in the model, but is defined concretely here.

Definition 3.1.1. *Given an instance of the independent cascade model, define the **state** of a node, u, at a given time, t, as $S(u,t) \in \{U, I, R\}$, where u is a node, t is the time step, and U signifies a node that has never been infected, I signifies a node that is infected and is live, and R signifies a node that has been infected but is no longer live.*

We will be referring to the probability that a given node u at time t is in a given state as **state probabilities**. These are expressed as $\Pr[S(u,t) \in \{I, U, R\}]$. These three possibilities are disjoint and every node is in one of these states at every time step. These probabilities will not simply be 0 or 1, but will vary, however, $\Pr[S(u,t) = I] + \Pr[S(u,t) = U] + \Pr[S(u,t) = R] = 1$.

The infection event in our model is also implicit, but is defined here so that we can determine the probability of this event occurring.

Definition 3.1.2. *Define $i(u,v,t)$ as the **infection event** where node u infects v at time t.*

Note that it is possible for two neighbors of v to infect v simultaneously in the same time-step. This is not a problem, because no matter the number of nodes that infect v simultaneously, the result is the same: v becomes infected.

We define an influence graph here as the informal structure for the graph on an instance of the independent cascade model.

Definition 3.1.3. *Given an instance of the independent cascade model on a graph $G = (V, E)$, the **influence graph** is a directed graph with vertex set*

equal to $V(G)$, and for every edge $\{u, v\} \in E(G)$ in the underlying graph, there are directed edges (u, v) and (v, u) with associated edge probabilities $b_{u,v}$ and $b_{v,u}$ as defined in subsection 2.1.2.

On the independent cascade model, the process of diffusion effectively traverses the influence graph, where nodes influence their neighbors based on the edge probabilities associated with the directed edges.

3.1.2 Preliminary Result

We first need to find the probability that any given node u, at any time step t, is uninfected, infected, or recovered. The following lemmas allow us to make the calculations necessary for computing for these probabilities. In this subsection we will provide a partial result on all graphs using the independent cascade model, but will expand upon this in the next section.

Proposition 3.1.4. *Given an instance of the independent cascade model, for all $u \in V$,*

$$\Pr[S(u,t) = I | S(u,t-1) = U] =$$
$$1 - \Pr\left[\bigwedge_{v \in N(u)} \neg\Big(i(v,u,t) \wedge (S(v,t-1) = I) \Big) \;\middle|\; S(u,t-1) = U \right]$$

Proof. This proposition says that the probability of node u becoming infected at time t given that it was not infected at time $t-1$ is equal to the probability that not all live, infected neighbors of u do not infect u. This is equivalent to saying, intuitively, that at least one live, infected neighbor of u infects u. Because of the order-independence property, the order in which the neighbors are chosen can be arbitrary.

Firstly, we can redefine the problem as the probability that an infection event occurs between any neighbor or u where the neighbor also is live and infected in time $t-1$:

$$\Pr[S(u,t) = I | S(u,t-1) = U] =$$
$$\Pr\left[\bigvee_{v \in N(u)} \Big(i(v,u,t) \wedge (S(v,t-1) = I) \Big) \;\middle|\; S(u,t-1) = U \right]$$

We rewrite this by taking its negation:

$$\Pr[S(u,t) = I | S(u,t-1) = U] =$$
$$1 - \Pr\left[\neg \bigvee_{v \in N(u)} \Big(i(v,u,t) \wedge (S(v,t-1) = I) \Big) \;\middle|\; S(u,t-1) = U \right]$$

Use De Morgan's laws to distribute the negation through the expression, changing the disjunction into a conjunction:

$$\Pr[S(u,t) = I | S(u,t-1) = U] =$$
$$1 - \Pr\left[\bigwedge_{v \in N(u)} \neg \Big(i(v,u,t) \wedge (S(v,t-1) = I) \Big) \;\middle|\; S(u,t-1) = U \right] \tag{3.1}$$

■

This result on the independent cascade model does not say too much in itself. We have shown that the probability that a node is infected can be formulated by observing the probabilities of its neighbors. We use this result to prove Proposition 3.2.4 in the following section.

3.2 Expected Influence on Acyclic Influence Graphs

In this section we provide a strong result on determining expected influence on acyclic influence graphs using the independent cascade model. We are able to determine the expected influence from any starting state in polynomial time on these types of graphs.

Definition 3.2.1. *An instance of a diffusion model on a graph G is defined as being on an **acyclic influence graph** if the subgraph formed by removing all 0 weight edges from the influence graph has no cycles of length greater than 2.*

This definition states that after a node has been infected, there is no path through which influence may even attempt to reinfect the node. Restricting our graph to this type allows us to show that two neighbors of any node both are independently infected and apply their probability of infection independently. Any cycles of length greater than 2 would cause this independence property to not hold.

We now have a stronger result for the probability of a node being infected at time t given that it was uninfected at time $t-1$. We will use this section to prove the following theorem:

Theorem 3.2.2. *The worst case running time of determining the expected influence of an instance of the independent cascade model on an acyclic influence graph, given an initial probability distribution of the states of every node at time $t = 0$, is $O\Big(|V(G)|^2 \cdot \max\{|V(G)|, |V(E)|\}\Big)$.*

This says that it is computationally feasible to determine the expected influence on an acyclic influence graph. The proof for this theorem will be provided further in this section. We first show how to achieve the formulation required to perform the appropriate expectation computation.

Lemma 3.2.3. *Given an instance of the independent cascade model with an acyclic influence graph, for all $u \in V$,*

1. $$\Pr[S(u,t) = I] = \Pr[S(u,t) = I \mid S(u,t-1) = U] \cdot \Pr[S(u,t-1) = U]$$

2. $\Pr[S(u,t) = R] = \Pr[S(u,t-1) = I] + \Pr[S(u,t-1) = R]$

3. $\Pr[S(u,t) = U] = 1 - \Pr[S(u,t) = R] - \Pr[S(u,t) = I]$

Proof. Covering these three cases:

1. By Bayes' Rule:
   $\Pr[S(u,t) = I] =$
   $$\frac{\Pr[S(u,t) = I \mid S(u,t-1) = U] \cdot \Pr[S(u,t-1) = U]}{\Pr[S(u,t-1) = U \mid S(u,t) = I]},$$
   but $\Pr[S(u,t-1) = U \mid S(u,t) = I] = 1$ because there is no way for a node to be active were it not uninfected in the previous time step.

2. Using the result from 1, a node is recovered if it was either infected or recovered in the previous step.

3. Using the results from 1 and 2, a node is uninfected if it is neither recovered nor infected.

■

We now show that it is actually possible to formulate the required values of Lemma 3.2.3.

Proposition 3.2.4. *Given an instance of the independent cascade model with an acyclic influence graph, for all $u \in V$,*

$$\Pr[S(u,t) = I \mid S(u,t-1) = U] = 1 - \prod_{v \in N(u)} \left(1 - \Pr[S(v,t-1) = I \mid S(u,t-1) = U] \cdot (b_{v,u})\right)$$

Proof. Following off of equation (3.1), we now see that both the probability of each infection event occurring is independent and the likelihood of any neighbor being infected is independent of any other neighbor being infected. This is clear because for one neighbor to affect the other, there would need

to be a path on the influence graph that connects one neighbor to the other through node u, but this is impossible because we have an acyclic influence graph. We can represent equation (3.1) as a product of probabilities where each neighbor does not infect u.

$$\Pr[S(u,t) = I | S(u,t-1) = U] =$$
$$1 - \prod_{v \in N(u)} \Big(1 - \Pr\left[i(v,u,t) \wedge (S(v,t-1) = I) \mid S(u,t-1) = U\right]\Big)$$

We can then re-write this equation by replacing the probability of infection event by the edge probability from the influence graph, since the probability of the infection event occurring if the node is active is exactly equal to the edge probability.

$$\Pr[S(u,t) = I \mid S(u,t-1) = U] =$$
$$1 - \prod_{v \in N(u)} \Big(1 - \Pr[S(v,t-1) = I \mid S(u,t-1) = U] \cdot (b_{v,u})\Big)$$

■

This result gives us a formulation for $\Pr[S(u,t) = I \mid S(u,t-1) = U]$, but it is still necessary to find a formulation for $\Pr[S(v,t) = I \mid S(u,t) = U]$.

Proposition 3.2.5. *Given an instance of the independent cascade model with an acyclic influence graph, for all $v \in V$, and for all $u \in N(v)$,*

$$\Pr[S(v,t) = I \mid S(u,t) = U] =$$
$$1 - \Pr\Bigg[\prod_{w \in N(v) - \{u\}} \Big(1 - \Pr[S(w,t) = I \mid S(v,t) = U] \cdot (b_{v,u}) \cdot \Pr[S(v,t) = U]\Big)\Bigg]$$

Proof. This proof follows all of the logic of equation (3.1) and Proposition 3.2.4.

First, represent $\Pr[S(v,t) = I \mid S(u,t) = U]$ as the probability that one neighbor of v that is not u infects v. That is to say, the probability that one neighbor w of v that is not u infects v at time t and w is infected at time $t-1$ and v is uninfected at time $t-1$.

$$\Pr[S(v,t) = I \mid S(u,t) = U] =$$
$$\Pr\left[\bigvee_{w \in N(v) - \{u\}} \Big(i(w,v,t) \wedge (S(w,t-1) = I) \wedge (S(v,t-1) = U)\Big)\right]$$

We rewrite this by taking its negation:

$$\Pr[S(v,t) = I \mid S(u,t) = U] =$$
$$1 - \Pr\left[\neg \bigvee_{w \in N(v) - \{u\}} \Big(i(w,v,t) \wedge (S(w,t-1) = I) \wedge (S(v,t-1) = U)\Big)\right]$$

Use De Morgan's laws to distribute the negation through the expression, changing the disjunction into a conjunction:

$$\Pr[S(v,t) = I | S(u,t) = U] =$$
$$1 - \Pr\left[\bigwedge_{w \in N(v) - \{u\}} \neg\Big(i(w,v,t) \wedge (S(w,t-1) = I) \wedge (S(v,t-1) = U)\Big)\right]$$

We can re-write this equation as a product of probabilities as in the proof of Proposition 3.2.4.

$$\Pr[S(v,t) = I | S(u,t) = U] =$$
$$1 - \prod_{w \in N(v) - \{u\}} \Big(1 - \Pr \Big[i(w,v,t) \wedge (S(w,t-1) = I) \wedge (S(v,t-1) = U) \Big] \Big)$$

We can rewrite the conjunctions by the multiplication rule of conditional probabilities.

$$\Pr[S(v,t) = I | S(u,t) = U] =$$
$$1 - \prod_{w \in N(v) - \{u\}} \Big(1 - \Pr \Big[(S(v,t-1) = U) \cdot \big(i(w,v,t) \wedge (S(w,t-1) = I) \mid S(v,t-1) = U \big) \Big] \Big)$$

We can then re-write this equation by replacing the probability of infection event by the edge probability from the influence graph, since the probability of the infection event occurring if the node is active is exactly equal to the edge probability.

$$\Pr[S(v,t) = I \mid S(u,t) = U] =$$
$$1 - \Pr \Bigg[\prod_{w \in N(v) - \{u\}} \Big(1 - \Pr[S(w,t-1) = I \mid S(v,t-1) = U] \cdot (b_{w,v}) \cdot \Pr[S(v,t-1) = U] \Big] \Bigg)$$

■

Here we see that the formulation for $\Pr[S(v,t) = I \mid S(u,t) = U]$ that

includes $\Pr[S(w,t-1) = I \mid S(v,t-1) = U]$. This recursive definition for $\Pr[S(v,t) = I \mid S(u,t) = U]$ allows us to calculate every value of $\Pr[S(v,t) = I \mid S(u,t) = U]$ for every v, every u, and every t.

Proposition 3.2.6. *Given an instance of the independent cascade model with an acyclic influence graph and a probability distribution of the states of every node at time $t = 0$, for every pair of neighbors $v, u \in V$, we can compute $\Pr[S(v,t) = I \mid S(u,t) = U]$ for every t.*

Proof. By induction over t:

Base Case For $t = 0$,

$$\Pr[S(v,0) = I \mid S(u,0) = U] = \begin{cases} 1 & : v \text{ is initially selected} \\ 0 & : v \text{ is not initially selected} \end{cases}$$

Inductive Case If we are able to determine $\Pr[S(w,t-1) = I \mid S(v,t-1) = U]$ and $\Pr[S(v,t-1) = U]$, then we are able to compute $\Pr[S(v,t) = I \mid S(u,t) = U]$ by Proposition 3.2.5.

We are able to determine $\Pr[S(u,t-1) = U]$ by Lemma 3.2.3 and we are able to compute $\Pr[S(w,t-1) = I \mid S(v,t-1) = U]$ by Proposition 3.2.5.

Since we have these two probabilities at time $t = 0$ and for all t, then we are able to determine the probability of $\Pr[S(w,t-1) = I \mid S(v,t-1) = U]$ for each pair of neighbors $v, u \in V$ at each time step. ∎

3.2.1 Main Theorems

We are able to use these calculations to show that we are able to determine the state of any node at any time step given an initial probability distribution for nodes in any of the three states. That is to say, if given the initial probability that each node is in any state, then we are able to determine the probability that the node will be in any state at any time step.

Lemma 3.2.7. *Given an instance of the independent cascade model with an acyclic influence graph and a probability distribution of the states of every node at time $t = 0$, for all $u \in V$, we can compute the probability that u is in any state at any time step.*

Proof. By induction over t:

Base Case For every node $u \in V$, we have $\Pr[S(u, 0) = I]$, $\Pr[S(u, 0) = U]$, and $\Pr[S(u, 0) = R]$ by definition.

Inductive Case If we are able to determine $\Pr[S(u, t-1) = I]$ and $\Pr[S(u, t-1) = U]$, then we are able to compute $\Pr[S(u, t) = I]$ by Proposition 3.1.4, and are able to determine $\Pr[S(u, t) = U]$ and $\Pr[S(u, t) = R]$ by Lemma 3.2.3.

Since we have the probabilities of the states at time $t = 0$ and for all t, then we are able to determine the probabilities of each state for each node at each time step. ■

For our work, we will be working primarily with an initial probability distribution where, for all $u \in V$:

$$\Pr[S(u, 0) = I] = \begin{cases} 1 & : u \text{ is initially selected} \\ 0 & : u \text{ is not initially selected} \end{cases}$$

$$\Pr[S(u, 0) = U] = \begin{cases} 0 & : u \text{ is initially selected} \\ 1 & : u \text{ is not initially selected} \end{cases}$$

The probability of the recovered state for each node is 0.

With Lemma 3.2.7, we are able to then determine the expected influence on any acyclic influence graph, given an initial probability distribution.

Statement 3.2.8. *Given an instance of the independent cascade model with an acyclic influence graph and a probability distribution of the states of every node at time $t = 0$, we are able to determine the expected influence for this graph at any arbitrary time $t = n$: $\sum_{u \in V} S(u, n) \in \{R, I\}$*

Algorithms

Algorithm 1: StateProbabilities(A)

input : independent cascade model with acyclic influence graph on graph $G = (V, E)$,
initially selected nodes A

output: state probabilities $\Pr[S(v,t) \in \{I, U, R\}]$

begin

```
// Initialize all nodes.
```

for *each* $v \in V(G)$ **do**

if $v \in A$ **then**

$\Pr[S(v,0) = I] \leftarrow 1$

$\Pr[S(v,0) = U] \leftarrow 0$

else

$\Pr[S(v,0) = I] \leftarrow 0$

$\Pr[S(v,0) = U] \leftarrow 1$

end

$\Pr[S(v,0) = R] \leftarrow 0$

end

for $t \leftarrow 1$ to $|V(G)| + 1$ **do**

for *each* $v \in V(G)$ **do**

// First get $\Pr[S(u,t) = I \mid S(u,t-1) = U]$.

tempI $\leftarrow 1$

for *each* $u \in N(v)$ **do**

// $\Pr[S(v,t) = I \mid S(u,t) = U]$ is computed below in Procedure ComputedConditional

tempI $\leftarrow$ tempI $\cdot \big(1 - (b_{v,u}) \cdot \Pr[S(v,t-1) = I \mid S(u,t-1) = U]\big)$

end

tempI $\leftarrow 1 -$ tempI

// Then set each of the probabilities.

$\Pr[S(v,t) = I] \leftarrow$ tempI $\cdot \Pr[S(v,t-1) = U]$

$\Pr[S(v,t) = R] \leftarrow \Pr[S(v,t-1) = I] + \Pr[S(v,t-1) = R]$

$\Pr[S(v,t) = U] \leftarrow 1 - \Pr[S(v,t) = R] - \Pr[S(v,t) = I]$

end

end

end

The following procedure shows the method for computing the values $\Pr[S(v,t) = I \mid S(u,t) = U]$ for every pair of neighbors $v, u \in V$, for time t. These will be stored and reused in this procedure and are used for the algorithm StateProbabilities.

Procedure ComputedConditional(A, t)

input : independent cascade model with acyclic influence graph on graph $G = (V, E)$,
initially selected nodes A, time t
output: probabilities $\Pr[S(v,t) = I \mid S(u,t) = U]$
begin
  for *each directed edge* (v, u) *in the influence graph of* G **do**
    `// Base case.`
    if $t = 0$ **then**
      if $v \in A$ **then**
        $\Pr[S(v,0) = I \mid S(u,0) = U] = 1$
      else
        $\Pr[S(v,0) = I \mid S(u,0) = U] = 0$
      end
    else
      temp $\leftarrow 1$
      for *each* $w \in N(v) - \{u\}$ **do**
        `// We assume that` $\Pr[S(v,t-1) = U]$ `is already computed. See StateProbabilities.`
        temp $\leftarrow$ temp $\cdot \big(1 - (b_{w,v}) \cdot \Pr[S(v,t-1) = U] \cdot \Pr[S(w,t-1) = I \mid S(v,t-1) = U]\big)$
      end
      temp $\leftarrow 1-$ temp
    end
  end
  $\Pr[S(v,t) = I \mid S(u,t) = U] = temp$
end

There are at most twice the number of edges of these probabilities to compute for every time step. This is because we need to generate these probabilities for every pair of nodes that are neighbors.

In StateProbabilities(A) we determine the state probabilities for every node. We then use the following algorithm to determine $\sigma(A)$ on the independent cascade model.

Algorithm 2: $\sigma(A)$

input : independent cascade model with acyclic influence graph on graph $G = (V, E)$,
initially selected nodes A
output: expected influence i
begin
  `// Run StateProbabilities to get all state probabilities up to time` $|V(G)| + 1$
  StateProbabilities(A)
  $n \leftarrow |V(G)|$
  for *each* $v \in V(G)$ **do**
    $i \leftarrow i + \Pr[S(v, n+1) = I] + \Pr[S(v, n+1) = R]$
  end
end

3.2.2 Theoretical Results

Proof of Theorem 3.2.2. Determining the expected influence as per statement 3.2.8 requires that we know the probability that each node is in infected or recovered steps at the final time step n. To attain the recovered and infected states at the final time step n, find the states of each node at time $n - 1$, as per Lemma 3.2.3.

There are at most $|V(G)|$ time steps in the independent cascade model and there are clearly $|V(G)|$ nodes to determine 3 states. Therefore, at most $3|V(G)|^2$ state calculations need to be performed.

To determine the infected state probability for a node, it is necessary to determine $\Pr[S(u, t) = I \mid S(u, t-1) = U]$, whose calculation performs a number of multiplications equal to the number of neighbors the node has. In the worst case of a completely connected graph, there are $|E(G)|$ multiplications, but there are always at least $|V(G)|$ calculations to perform over

the graph. Therefore for each of the time steps and each of the nodes we need to perform $\max\{|V(G)|, |V(E)|\}$ multiplications.

Each of these multiplications also requires that all values for $\Pr[S(v,t) = I \mid S(u,t) = U]$ are computed. There are at most twice as many of these calculations as edges, and $|V(G)|$ time steps which to calculate these, so this adds $|V(G)| \cdot 2|V(E)|$ to the running time. These values are computed in-time, and are not computed every time for different nodes, so this does not multiply a factor to the running time.

The worst case running time is at most $3|V(G)|^2 \cdot \max\{|V(G)|, |V(E)|\} + |V(G)| \cdot |V(E)|$, but the last term drops out since it can never be greater than $|V(G)|^2 \cdot \max\{|V(G)|, |V(E)|\}$, so the final worst case running time is $\mathrm{O}\Big(|V(G)|^2 \cdot \max\{|V(G)|, |V(E)|\}\Big)$ ■

3.3 Influence Maximization on Paths

We have also determined that it is possible, on the independent cascade model of a path, to determine the optimal selection of initial nodes that maximizes expected influence. We achieve this through a dynamic programming approach where we build up a table of optimal results for smaller subsets of the problem that are then used to compute the optimal values for the superset.

3.3.1 Definitions

Definition 3.3.1. *A **path** is a graph that can be sequenced in such a way that there are edges between any two adjacently sequenced nodes and no other edges.*

Label the nodes of this path from 1 to $|V(P)|$, representing the number of nodes in the path. This labeling follows the sequence defined above. Call P_n a subpath of P containing the nodes labeled from 1 to n. We may also write this as, $\{1, 2, \ldots, n-1, n\}$. P_n represents an instance of the independent cascade model containing said subpath of P. Call $P_{j,i}$ a subpath

of P containing the nodes labeled from j to i or $\{j, \ldots, i\}$. Call S the number of initially selected nodes in an instance of the following decision problem.

We have shown previously that is possible to determine the expected influence on an acyclic influence graph. Denote $\sigma(i)$ as the expected influence on the subpath P_i where only the node labeled i is initially selected. Denote $\sigma(j, i)$ as the expected influence on the subpath $P_{j,i}$ where only the nodes labeled i and j are initially selected. Denote $\sigma'(j, i)$ as the expected influence on the subpath $P_{j,i}$ where only the node labeled j is initially selected.

$\hat{\sigma}(i, k)$ is the maximum expected influence function that we will be using to maximize over the path. $\hat{\sigma}(i, k)$ represents the maximum expected influence on the subpath P_i if exactly k nodes are initially selected including the node labeled i. This maximum expected influence differs from σ in that $\hat{\sigma}$ represents the expected influence from the best possible selection of nodes.

We define $\hat{\sigma}(i, k)$ recursively in the following way:

$$\hat{\sigma}(i,k) = \begin{cases} 0 & : k = 0 \\ \sigma(i) & : k = 1 \\ \max\limits_{j \in (k-1, \ldots, i-1)} \hat{\sigma}(j, k-1) + \sigma(j, i) - 1 & : otherwise \end{cases}$$

To maximize influence on P with S initial nodes, calculate the following, where $n = |V(P)|$:

$$\max_{j \in (1, \ldots, n-1)} (\max\{\hat{\sigma}(j, S-1) + \sigma(j, n) - 1, \hat{\sigma}(j, S) + \sigma'(j, n) - 1\})$$

This is a necessary in addition to $\hat{\sigma}$ because it allows for the possibility that the last node in the path is not selected.

3.3.2 Algorithm

The following algorithm optimally maximizes influence on paths.

Algorithm 3: MaxPathInfluence(P, K)

input : independent cascade model on path $P = (V, E)$,
number of selected nodes K
output: maximum expected influence F,
optimal selection $\hat{S}$

begin

for $i \leftarrow 1$ *to* $|V(P)|$ **do**

`// Base cases.`

$\hat{\sigma}(i, 0) \leftarrow 0$

$\hat{S}(i, 0) \leftarrow \emptyset$

$\hat{\sigma}(i, 1) \leftarrow \sigma(i)$

$\hat{S}(i, 1) \leftarrow \{i\}$

for $k \leftarrow 2$ *to* $min(i, K)$ **do**

$\text{temp}\hat{S} \leftarrow \underset{j \in \{1,\ldots,i-1\}}{\arg\max} (\hat{\sigma}(j, k-1) + \sigma(j, i)$

$\hat{\sigma}(i, k) \leftarrow \hat{\sigma}(\text{temp}\hat{S}, k-1) + \sigma(j, i) - 1$

$\hat{S}(i, k) \leftarrow \hat{S}(\text{temp}\hat{S}, k-1) \cup \{i\}$

end

end

$\text{temp}\hat{S} \leftarrow \underset{j \in \{1,\ldots,i-1\}}{\arg\max} (\max\{\hat{\sigma}(j, K-1) + \sigma(j, i), \hat{\sigma}(j, K) + \sigma'(j, i)\})$

`// If we select the last node in the path, add it to the optimal set.`

if $\hat{\sigma}(temp\hat{S}, K-1) + \sigma(j, i) > \hat{\sigma}(temp\hat{S}, K) + \sigma'(j, i)$ **then**

$\hat{S} \leftarrow \hat{S}(\text{temp}\hat{S}, K-1) \cup \{i\}$

else

$\hat{S} \leftarrow \hat{S}(\text{temp}\hat{S}, K)$

end

$F \leftarrow \max\{\hat{\sigma}(\text{temp}\hat{S}, K-1) + \sigma(j, i) - 1, \hat{\sigma}(j, K) + \sigma'(\text{temp}\hat{S}, i) - 1\}$

end

3.3.3 Theoretical Results

We will now show that this algorithm provides correct results and runs in polynomial time.

Theorem 3.3.2. *The algorithm MaxPathInfluence produces the maximum expected value with K selected nodes on the independent cascade model with underlying graph being a path, P.*

We can see that each entry $\hat{\sigma}(i,k)$ in the dynamic programming table represents the optimal solution for the path up to length i using k selected nodes where the ith node is necessarily selected. We can see that each entry in the table is composed of a smaller entry, which represents the best possible position to place another selected node. $\sigma(j,i)$ gives us the influence produced in the remainder of the path, that is, the part of the path that has no more selected nodes.

The algorithm gives us divisions of the path that optimally pick the "furthest" node away from vertex 1, not including the ith node. From a top-down view, we can see that we choose a node, calculate the influence on one side of that node and then split up the rest of the path with the remainder of the allowable nodes.

We will first prove by induction on k that the function $\hat{\sigma}(i,k)$ returns the maximum expected value on P_i where k nodes are selected including the ith node for all k and for all i.

Lemma 3.3.3. *The function $\hat{\sigma}(i,k)$ returns the maximum expected value on P_i where k nodes are selected including the ith node for all k and for all i.*

Proof. By induction on k:

Base Case $(k = 0)$: For all i, when k is 0, the maximum expected value is 0.

$(k = 1)$: For all i, when k is 1, the maximum expected value is equivalent to $\sigma(i)$, which is maximum by the definition of $\sigma(i)$.

Inductive Step If we assume that for all $l < i$ that $\hat{\sigma}(l, k-1)$ produces the maximum expected value in its own case, then there must be some node $k-1 \leq j < i$ for which $\hat{\sigma}(j, k-1)+\sigma(j, i)$ is maximized and this produces the maximum value for $\hat{\sigma}(i, k)$.

Because we check every possible value, one possible value for j must maximize the entire remainder of the of the path.

■

We now return to proving Theorem 3.3.2:

Proof of Theorem 3.3.2. Following Lemma 3.3.3, we can see that, similarly, there is a node $s \in \{k, \ldots, |V(P)|\}$ such that either $\hat{\sigma}(j, s-1)+\sigma(j, n)-1$ (if the nth node is selected) or $\hat{\sigma}(j, s) + \sigma'(j, n) - 1$ (if the nth node is not selected) that maximizes the expected influence over the entire path P.

We know that we can maximize the superproblem because in both parts of this formulation we are testing every possible subproblem. ■

Theorem 3.3.4. *The worst case running time of the algorithm MaxPathInfluence on an instance of the independent cascade model with an underlying path, P is $O\big(|V(P)|^5\big)$.*

Proof. The worst case running time of $\sigma(j, i)$ and $\sigma(i)$ and $\sigma'(j, i)$, which we shall collectively call $\sigma(\cdot)$ is at most $\mathrm{O}(|V(P)|^2)$ by virtue of each of these functions having at most two elements selected on the ends of their sub-paths. To calculate expected influence we make at most $\big(|V(P)|^2\big)$ calculations. Each calculation of $\hat{\sigma}(i, k)$ requires at most i calculations of $\sigma(\cdot)$, and we must find at most $(i \cdot k)$ elements in the $\hat{\sigma}(i, k)$ table. Both i and k must be at most $|V(P)|$, so we have $\mathrm{O}(|V(P)|^2)$ calculations that each perform $\mathrm{O}\big(|V(P)|\big)$ calculations of the function that takes $\mathrm{O}\big(|V(P)|^2\big)$ time, so the worst case running time is $\mathrm{O}\big(|V(P)|^5\big)$. ■

3.4 Influence Maximization on Trees

Our research has not established an efficient way to determine the maximum expected influence on any arbitrary tree. We have, in lieu of this, determined a polynomial time approximation algorithm for this problem. This result is achieved by creating an exponential time solution and then rounding the possible options into a polynomial number of boxes. Like in the algorithm for paths, we will be using a dynamic programming solution to generate the answers to subproblems of the original tree.

We will begin this section referring only to binary trees, but will provide a method for extending our results to arbitrary trees in Subsection 3.4.6.

3.4.1 Definitions

We will be executing our solution on an instance of the independent cascade model with underlying graph being a tree, T.

Definition 3.4.1. *A **tree** is a connected graph without cycles.*

This means that there is exactly one path between all pairs of vertices in the graph. This is different from the acyclic influence graph defined previously. The tree we describe here refers specifically to the graph provided to the diffusion model, and does not contain information on the edge probabilities at all. The acyclic influence graph can be any graph where there cannot be a cycle over non-zero edge probabilities. A diffusion model with an underlying tree graph can represent an acyclic influence graph, but the converse is not necessarily true.

As outlined in the background section, we will define $\sigma(A)$ in terms of another function, $\varphi_G(S)$. For a graph $G = (V, E)$ and a set $S \subseteq V$, define $\varphi_G(S)$ as the number of nodes infected by the end of diffusion if the nodes in S are the only nodes in V to be initially selected. Here, $\varphi_G(S)$ is a random function that we will use to base σ. Define $\sigma_G(S)$ as the expected value on G of $\varphi_G(S)$: $\sigma_G(S) =_{\text{def}} \mathbb{E}[|\varphi_G(S)|]$.

For a subgraph H of G, define $\varphi_{H,G}(S)$ as the number of nodes in H that are infected by the end of the process. Similarly, define $\sigma_{H,G}(S) =_{\text{def}}$

$\mathbb{E}[|\varphi_{H,G}(S)|]$. This terminology allows us to relate the expected number of nodes infected in a graph and its subgraph.

3.4.2 Method

Allow $T = G$ to be an rooted tree, where the root is arbitrary. Viewing the graph as a rooted tree allows us to treat each neighbor as a child of T and use the root as a reference point. For any node $r \in T$, let T_r be the subtree of T rooted at r. Call $(r, \emptyset)$ the subgraph containing just the node r and no edges.

For a tree rooted at r, let a and b be the two children of r. For any subset $S \subseteq T_r$, we find that

$$\varphi_{T_r}(S) = \varphi_{(r,\emptyset),T_r}(S) + \varphi_{T_a,T_r}(S) + \varphi_{T_b,T_r}(S)$$

This holds because the number of infected nodes in the rooted tree must be the number of infected nodes in both of its children's trees and the root.

Following this, we can show by linearity of expectation that

$$\begin{aligned}
\sigma_{T_r}(S) &= \mathbb{E}[|\varphi_{T_r}(S)|] \\
&= \mathbb{E}[|\varphi_{(r,\emptyset),T_r}(S)| + |\varphi_{T_a,T_r}(S)| + |\varphi_{T_b,T_r}(S)|] \\
&= \mathbb{E}[|\varphi_{(r,\emptyset),T_r}(S)|] + \mathbb{E}[|\varphi_{T_a,T_r}(S)|] + \mathbb{E}[|\varphi_{T_b,T_r}(S)|] \\
&= \sigma_{(r,\emptyset),T_r}(S) + \sigma_{T_a,T_r}(S) + \sigma_{T_b,T_r}(S)
\end{aligned}$$

This states that we can solve σ for the root and each of its subtrees separately. We will show that by computing each part of $\sigma_{T_r}(S)$ successively, we can divide and conquer the entire tree.

Let A be the subset of vertices in S that are in T_a, or $A = S \cap T_a$. Similarly, $B = S \cap T_b$. Let $p_a(A)$ be the probability that a is infected given that we initialize only the nodes in A in T_a. What we mean by $p_a(A)$ is the probability that a is infected "from below" by influence from its children. This does not include the probability that a is infected through a's parent or any other part of the tree. Define $p_b(B)$ the same way for T_b. We will be able to determine these probabilities recursively as will be seen below when we define our dynamic programming. We will now, however, define the σ

functions that will be crucial in actually executing the dynamic program.

Firstly,

$$\sigma_{(r,\emptyset),T_r}(S) = \begin{cases} 1 & : \text{if } r \in S \\ w_{a,r} \cdot p_a(A) + w_{b,r} \cdot p_b(B) - & \\ w_{a,r} \cdot p_a(A) \cdot w_{b,r} \cdot p_b(B) & : \text{otherwise,} \end{cases}$$

where each $w_{u,v}$ represents the independent cascade model activation weights from node u to node v.

The logic of this is that the expected number of infected nodes in just the root is either 1 if the root is selected or it is equal to the probability that either or both of the children of r infect r.

Further, we look at $\sigma_{T_a,T_r}(S)$ and $\sigma_{T_b,T_r}(S)$, but because the logic is equivalent, we will write out just the method for determining $\sigma_{T_a,T_r}(S)$ with the understanding that $\sigma_{T_b,T_r}(S)$ is determined by switching all as to bs, and so on.

The expected number of nodes infected in T_a depends on its root, r. Either r influences a by virtue of being part of the initially selected set, S, or r influences a by first being infected in the subtree rooted at b. We assume that when r is active, it influences a with probability $w_{r,a}$ even if a is already active because the effect of influence from r is subsumed by a already being in the active list. We derive the result:

$$\sigma_{T_a,T_r}(S) = \begin{cases} (1 - w_{r,a}) \cdot \sigma_{T_a}(A) + w_{r,a} \cdot \sigma_{T_a}(A \cup \{a\}) & : \text{if } r \in S \\ \big(1 - w_{b,r} \cdot w_{r,a} \cdot p_b(B)\big) \cdot \sigma_{T_a}(A) + & \\ \big(w_{b,r} \cdot w_{r,a} \cdot p_b(B)\big) \cdot \sigma_{T_a}(A \cup \{a\}) & : \text{otherwise.} \end{cases}$$

3.4.3 Algorithm

From these results, we can maximize σ_{T_r}, recursively, by saving, for each $k < K$, the set S of size k corresponding to the unique triple $(k, \sigma_{T_r}(S), \sigma_{T_r}(S \cup \{r\}))$ that maximizes $p_r(S)$. To prevent confusion between the function $\sigma_{T_r}(S)$ and its indexed value, we will let $x = \sigma_{T_r}(S)$ and $y = \sigma_{T_r}(S \cup \{r\})$. Therefore, our unique triple is (k, x, y).

Our dynamic program follows directly from the definitions of $\sigma_{(r,\emptyset),T_r}(S)$ and $\sigma_{T_a,T_r}(S)$. We have defined $\hat{S}_r(k, x, y)$ as the optimal selection of k

nodes where $\sigma_{T_r}(\hat{S}) = x$ and $\sigma_{T_r}(\hat{S} \cup \{r\}) = y$ that maximizes p_r. The exact value of p_r is listed as $p_r(k, x, y)$.

Our dynamic program will iterate over every possible candidate distribution of k initially selected nodes between subtrees T_a, T_b and the root r.

Let $C_r(k, x, y)$ be the set of all 6-tuple candidates $(k', x', y', k'', x'', y'')$ satisfying the following:

$$\begin{aligned} &k = k' + k'', \text{ where } k' \text{ and } k'' \text{ are natural numbers,} \\ &x = \sigma_{T_r}(\hat{S}_a(k', x', y') \cup \hat{S}_b(k'', x'', y'')), \text{ and} \\ &y = \sigma_{T_r}(\hat{S}_a(k', x', y') \cup \hat{S}_b(k'', x'', y'') \cup \{r\}). \end{aligned}$$

The set $C_r(k, x, y)$ contains all possible candidate distributions that do not include the root in the set of initially selected nodes.

Let $\hat{C}_r(k, x, y)$ denote a 6-tuple $(k', x', y', k'', x'', y'') \in C_r(k, x, y)$ that maximizes

$$\begin{aligned} &w_{ar} \cdot p_a(k', x', y') + w_{br} \cdot p_b(k'', x'', y'') - \\ &\qquad w_{ar} \cdot p_a(k', x', y') \cdot w_{br} \cdot p_b(k'', x'', y''). \end{aligned}$$

Let $C_r^1(k, x, y)$ be the set of all 6-tuples $(k', x', y', k'', x'', y'')$ of satisfying the following:

$$\begin{aligned} &k = k' + k'' + 1, \text{ where } k' \text{ and } k'' \text{ are natural numbers,} \\ &x = y = \sigma_{T_r}(\hat{S}_a(k', x', y') \cup \hat{S}_b(k'', x'', y'') \cup \{r\}). \end{aligned}$$

By comparing the maximized values, we are able to determine which candidate distribution maximizes p_r. We preferentially choose the candidate that chooses the root since this clearly maximizes the probability that the root is selected.

$$p_r(k,x,y) = \begin{cases} 1 & \text{if } C_r^1(k,x,y) \neq \emptyset, \\ \Big(w_{ar} \cdot p_a(k',x',y') + & \text{if } C_r(k,x,y) \neq \emptyset, \text{where} \\ \quad w_{br} \cdot p_b(k'',x'',y'') - & \quad (k',x',y',k'',x'',y'') = \\ \quad w_{ar} \cdot p_a(k',x',y') \cdot w_{br} \cdot & \quad \hat{C}_r(k,x,y), \\ \quad p_b(k'',x'',y'')\Big) & \\ 0 & \text{otherwise.} \end{cases}$$

$$\hat{S}_r(k,x,y) = \begin{cases} \hat{S}_a(k',x',y') \cup & \text{if } C_r^1(k,x,y) \neq \emptyset, \text{where} \\ \quad \hat{S}_b(k'',x'',y'') \cup \{r\} & \quad (k',x',y',k'',x'',y'') \in \\ & \quad C_r^1(k,x,y), \\ \hat{S}_a(k',x',y') \cup & \text{if } C_r(k,x,y) \neq \emptyset, \text{where} \\ \quad \hat{S}_b(k'',x'',y'') & \quad (k',x',y',k'',x'',y'') = \\ & \quad \hat{C}_r(k,x,y), \\ \emptyset & \text{otherwise.} \end{cases}$$

Since we know the optimal set to choose at each rooted subtree that maximizes the probability that the root is selected, we can begin from the furthest leaf and work our way up the tree maintaining the set that provides the best probability of each subroot being selected. This is the basis of a dynamic program for a recursive solution. At each step we will try every distribution of the k nodes between the two children and the root. This, along with saving every relevant table entry, allows us to efficiently work our way up the tree.

For our algorithm, we will assume a labeling of the nodes, $(1, \ldots, |V(T)|)$, where the root is labeled $|V(T)|$ and the leaf that is furthest away from the root is 1. The remaining nodes are labeled from the leaf to the root, where one layer of depth at a time is labeled before the next shallower layer is labeled. This way we can simply move up the tree by counting the labels. We will denote T_i as the subtree rooted at the label i. Call T_i^a the tree rooted at one child of i and T_i^b the tree rooted at the other child of i. If i has no children then T_i^a and T_i^b are $\emptyset$.

Algorithm 4: MaxBinaryTreeInfluence(T, K)

input : independent cascade model on tree $T = (V, E)$,
number of selected nodes K
output: maximum expected influence F, optimal selection O
begin
  for $r \leftarrow 1$ *to* $|V(T)|$ **do**
    for $k \leftarrow 0$ *to min*$\big(K, |V(T_r)|\big)$ **do**
      `// Base case.`
      if $k = 0$ **then**
        $p_r(0, 0, 0) \leftarrow 0$
        $\hat{S}_r(0, 0, 0) \leftarrow \emptyset$
      else
        get $C_r(k, x, y)$
        calculate $\hat{C}_r(k, x, y)$
        get ${C_r}^1(k, x, y)$
        if ${C_r}^1(k, x, y) \neq \emptyset$ **then**
          $p_r(k, x, y) = 1$
          $\hat{S}_r(k, x, y) = \hat{S}_a(k', x', y') \cup \hat{S}_b(k'', x'', y'') \cup \{r\}$,
            where $(k', x', y', k'', x'', y'') \in C_r^1(k, x, y)$
        else if $C_r(k, x, y) \neq \emptyset$ **then**
          $p_r(k, x, y) = w_{ar} \cdot p_a(k', x', y') + w_{br} \cdot p_b(k'', x'', y'') -$
            $w_{ar} \cdot p_a(k', x', y') \cdot w_{br} \cdot p_b(k'', x'', y'')$
          $\hat{S}_r(k, x, y) = \hat{S}_a(k', x', y') \cup \hat{S}_b(k'', x'', y'')$,
            where $(k', x', y', k'', x'', y'') \in C_r(k, x, y)$
        else
          $p_r(k, x, y) = 0$
          $\hat{S}_r(k, x, y) = \emptyset$
        end
      end
    end
  end
  $F \leftarrow \underset{\sigma_{T_{|V(T)|}}(S)}{\arg\max}\, \hat{S}_r(K, \sigma_{T_{|V(T)|}}(S), \sigma_{T_{|V(T)|}}(S \cup |V(T)|)$
end

3.4.4 Theoretical Results

Here we show that our method produces the correct value for the maximum expected influence on a binary tree.

Theorem 3.4.2. *The algorithm MaxBinaryTreeInfluence produces the maximum expected value with K selected nodes on the independent cascade model with underlying graph being a binary tree, T.*

Proof. Firstly, we will see that by generating all of the table entries, $\big(k, \sigma_{T_r}(S), \sigma_{T_r}(S \cup \{r\})\big)$, we will always maintain the maximum expected value at every step by induction on the depth, d, of the current node:

Base Case When the current node is a leaf and k = 0, the maximum expected value is 0.

When the current node is a leaf and k = 1, the maximum expected value is 1.

Inductive Step Assume that the maximum expected value for tree depth $d-1$ for all applicable k is known, then for every node r at depth d, iterate over all possible distribution candidates for all applicable k. Because we have every table entry for the children of i, we store $\hat{S}_r\big(k, \sigma_{T_r}(S), \sigma_{T_r}(S \cup \{r\})\big)$ table values for every partition. If we generate a table value that has already been computed, then we choose the set that produces the greatest value for $p_r\big(k, \sigma_{T_r}(S), \sigma_{T_r}(S \cup \{r\})\big)$. If all other things are equal, the set that produces the greatest probability of the root being active will therefore always have a greater expected value when the entire tree is considered, since the root will be more likely infect other nodes.

Since we are checking over all possible candidates and storing all relevant table values, we simply choose the set that produces the greatest possible expected influence for $\big(K, \sigma_{T_r}(S), \sigma_{T_r}(S \cup \{r\})\big)$ to attain our final solution. ∎

We now show that our binary tree algorithm runs in exponential time and that our approximation scheme produces polynomial time approximations to some error.

Theorem 3.4.3. *The worst case running time of the algorithm MaxBinaryTreeInfluence on an instance of the independent cascade model with an underlying binary tree, T is $O(2^{|V(T)|} \cdot |V(T)|^4)$.*

Proof. This dynamic program first loops over $|V(T)|$ nodes, then over all possible values of k, then partitions those k elements, then cycles through every relevant table entry. When there are distinct values, we can cycle over potentially $k2^{2|V(T)|}$ entries for $\big(k, \sigma_{T_r}(S), \sigma_{T_r}(S \cup \{r\})\big)$ that maximize $p_r(S)$. That leads to $k^3 \cdot |V(T)| \cdot 2^{2|V(T)|}$. Because k can be at most $|V(T)|$, the final running time in the worst case is $\mathrm{O}\big(2^{|V(T)|} \cdot |V(T)|^4\big)$.

■

3.4.5 Approximation

There are a potentially exponential number of table entries to compute because both $\sigma_{T_r}(S)$ and $\sigma_{T_r}(S \cup \{r\})$ may be rational numbers that can be distinct for each subset, S. Because of this, we cannot use this method to produce a polynomial time solution in the general case. Instead, we can use this algorithm to approximate the true value of the maximum expected influence on binary trees.

We accomplish this by rounding each of the two rational parameters above into bins of fixed size. The simplest example is to simply take the floor of the parameters, and have as table entries $\big(k, \lfloor \sigma_{T_r}(S) \rfloor, \lfloor \sigma_{T_r}(S \cup \{r\}) \rfloor\big)$. From this, there are at most $|V(T_r)|^2$ possible entries for each value of k because with integer values, there can only be at most $|V(T_R)|$ possible values for the expected influence.

The maximum absolute error for maximum expected influence is the number of nodes in the graph times the bin size. In the case of taking the floor, we have an error of $|V(T_R)|$. We therefore compromise the number of table entries by the error. For instance, if the bin size were $1/|V(T_r)|$ then

the error is at most 1, but we would have a maximum $k|V(T_r)|^4$ table entries. Thus, we can easily modify the algorithm so that every time we store our value for $\big(k, \sigma_{T_r}(S), \sigma_{T_r}(S \cup \{r\})\big)$, we round down the last two parameters to the nearest bin interval. For each σ, simply divide its value by the bin size, take the floor, and then multiply by the bin size. To approximate $\sigma_{T_r}(S)$ with bin size ε: $\sigma_{T_r}(S) = \lfloor \frac{\sigma_{T_r}(S)}{\varepsilon} \rfloor \cdot \varepsilon$.

Theorem 3.4.4. *The worst case running time of the algorithm MaxBinaryTreeInfluenceApproximation on an instance of the independent cascade model with an underlying binary tree, T, with bin size ε is $O\big(\frac{|V(T)|}{\varepsilon}^6\big)$.*

Proof. Following the previous proof, we see that we need to compute some number of table entries to generate our solution. The number of table entries is determined by the size of each of the bins. K can be at most $|V(T)|$ and there can be at most $\frac{|V(T)|}{\varepsilon}$ possible values for $\sigma_{T_r}(S)$ and $\sigma_{T_r}(S \cup \{r\})$ so there are at most $|V(T)| \cdot \big(\frac{|V(T)|}{\varepsilon}\big)^2$ table entries, yielding a worst case running time of $\mathrm{O}\big(\frac{|V(T)|}{\varepsilon}^6\big)$ following Theorem 3.4.3.

■

Theorem 3.4.5. *The algorithm MaxBinaryTreeInfluenceApproximation, with bin size ε, calculates the maximum expected influence on a binary tree with absolute error of at most $\varepsilon \cdot |V(T)|$.*

Proof. Proof by structural induction on subtrees:

Base Case On a leaf, the absolute error is at most $\varepsilon + c$ for some very small c because the most the approximation can differ from the actual expected value is $\varepsilon + c$.

Inductive Case If we assume that the approximation on subtrees T_a and T_b is m_a and m_b respectfully, then the expected influence is $m_a + m_b +$ the error from the root. The approximation on the root can differ from the actual expected value of the root by at most $\varepsilon + c$ for some very small c.

Because our approximation differs from the true expected value by ε for each node, the absolute error is at most $\varepsilon \cdot |V(T)|$. ■

3.4.6 Extension to Arbitrary Trees

We have thus far shown an exponential time solution on binary trees and a polynomial time approximation solution on binary trees. We can extend this finding to all arbitrary trees by creating dummy nodes whenever a vertex has more than two children. We will first reformat the tree to be a binary tree and then modify the algorithm slightly so that we can perform the same calculations with only minor modifications. This is done as follows.

Reformat Step First, we will reformat the tree at the labeling step. Before labeling, we will ensure that we have a binary tree. For each node, q, that has more than two children, create a new tree as follows:

1. Create a vertex q_z^1 with a subscript that we shall call z. Assign two of the children of q to be children of q_z^1.
2. Then, for each remaining child of q create a new vertex, say q_z^i and assign as its children q_z^{i-1} and one remaining child of q.
3. The edge weights between each q_z^i and the children of q are set to whatever the edge weights were between q and its children.
4. The edge weights between any two nodes q_z^i and q_z^{i-1} are set to 1. This means that the probability that one of these nodes is activated is the probability that any of these nodes are activated.

We can then remove the superscripts from each q_z and then label the tree as normal, while still maintaining the subscript flag z, which we will use in modifying the tree algorithm.

Note that there will be at most $2|V(T)| - 2$ nodes in the tree if there is one node that has every other node as child. In this case we will need to create $|V(T)| - 2$ extra nodes.

We shall call this new tree a **binary modified arbitrary tree**.

Algorithm Step Our algorithm changes in only two specific ways. Firstly, when looping over all possible the number of initially selected nodes, k, do not count any flagged nodes, and instead count 1 for each different flag in that subtree. This way we are not allowing more than one of the flagged nodes to "use up" one of the allowed initially selected nodes. Secondly, when computing the value $\sigma_{(r_z,\emptyset),T_r}$, set it to 1 only if $r_z \in S$ and only if there are no other nodes in the subtree rooted at r_z with flag z. This means that when we calculate the influence granted by one of our flagged nodes, we only add in its influence once.

Stated formally, the recursive definition of $\hat{p}_r(k, x, y)$ changes as such. For some node r_z that has a flag.

$$p_r(k,x,y) = \begin{cases} 1 & \text{if } C_r^1(k,x,y) \neq \emptyset \\ & \quad \text{and } r_z \text{ has no children} \\ & \quad \text{flagged with } z, \\ \Big(w_{ar} \cdot p_a(k',x',y') + & \text{if } C_r(k,x,y) \neq \emptyset, \text{ where} \\ \quad w_{br} \cdot p_b(k'',x'',y'') - & \quad (k',x',y',k'',x'',y'') = \\ \quad w_{ar} \cdot p_a(k',x',y') \cdot w_{br} \cdot & \quad \hat{C}_r(k,x,y), \\ \quad p_b(k'',x'',y'')\Big) & \\ 0 & \text{otherwise.} \end{cases}$$

The algorithm for determining the maximum expected influence changes MaxBinaryTreeInfluence like so:

Algorithm 5: MaxTreeInfluence(T, K)

input : independent cascade model on tree $T = (V, E)$,
number of selected nodes K
output: maximum expected influence F, optimal selection O
begin
`// Change the condition` $C_r^1(k, x, y) \neq \emptyset$ `to the following:`
if $C_r^1(k, x, y) \neq \emptyset$ *and (*r *has flag* z *and* r_z *has no children flagged with* z*) or* r *has no flag* **then**
⋮
end
⋮
remainder of function continues as before
⋮
end

We do not need to modify the remainder of the algorithm because we have already hard-coded to not double count any flagged nodes as far as k is concerned. These two steps are enough to allow us to compute our solution on any tree using the framework provided by our binary tree algorithm. We have ensured that we will not count any of the new nodes more than once. We will have more nodes in our table, because we have an extra row for each child above two in all nodes with more than two children. However, we add at most $|V(T)|-2$ extra nodes (if one node has every other node as child), and this will not increase the computation required, as we already have an exponential time solution and this would simply add another multiplier, which is subsumed in the greater complexity of the process as a whole.

Theorem 3.4.6. *The algorithm MaxTreeInfluence produces the maximum expected value with K selected nodes on the independent cascade model with underlying graph being a binary modified arbitrary tree, T.*

It is enough to show that given a subtree T_r that has more than 2 nodes that the binary modified subtree T_{r_z} will count the maximum expected influence correctly. That is to say, if one flagged node computes itself correctly,

then we can apply these results to the entire tree, because the rest of the tree will form a binary tree, which is computable.

Proof. We start the proof by induction on the depth, d, of the flagged node. As a reminder, the deepest node has depth $d = 1$.

Base Case Firstly, the tree rooted at deepest node $T^1_{r_z}$ properly determines the influence of its two children, because this is simply following the algorithm for binary trees.

Inductive Step Assume that the node at depth $d-1$ properly determines the maximum expected influence, we try to determine the maximum influence on node at depth d or the tree rooted $T^d_{r_z}$. The influence from the one child, c, of r^d_z to the flagged child, f_z is exactly equal to the weight of the edge between ${r_z}^d$ and c or w_{c,r^d_z}. Likewise, the influence from f_z to c is $w_{r^d_z,c}$. This is because the weight of the edge between the two flags is 1. Because of this, and because of the same linearity of expectations we used above, we can factor in the influence from the deepest node's children to any other node along the chain of flagged nodes in exactly the same way we computed the binary tree influence definition. Therefore, we have the maximum expected influence by simply following previous logic. ■

Theorem 3.4.7. *The worst case running time of the algorithm MaxTreeInfluence on an instance of the independent cascade model with an underlying binary modified arbitrary tree, T, is $O\big(2^{4|V(T)|} \cdot |V(T)|^4\big)$.*

Proof. It is a trivial process to modify an arbitrary tree into a binary tree and adds only a constant to the computation equal to the number of created nodes, which is at worst $|V(T)| - 2$. Further, since we are adding $|V(T)| - 2$ nodes, we are effectively doubling the number of nodes to compute and squaring the number of table entries. Note that k does not change by our process and remains bounded at $|V(T)|$. Returning to Theorem 3.4.3, this dynamic program first loops over $2|V(T)|$ nodes, then over all

possible values of k, then partitions those k elements then cycles through every relevant table entry. When there are distinct values we can cycle over potentially $k2^{4|V(T)|}$ entries for $\big(k, \sigma_{T_r}(S), \sigma_{T_r}(S \cup \{r\})\big)$ that maximize $p_r(S)$. That leads to $\mathrm{O}\big(k^3 \cdot 2|V(T)| \cdot 2^{4|V(T)|}\big)$. Because k can be at most $|V(T)|$, the final running time in the worst case is $\mathrm{O}\big(2^{4|V(T)|} \cdot 2|V(T)|^4\big)$ or $\mathrm{O}\big(2^{4|V(T)|} \cdot |V(T)|^4\big)$ ■

We execute our approximation algorithm exactly the same way for the binary modified arbitrary tree as we did for the binary tree.

Theorem 3.4.8. *The worst case running time of the algorithm MaxBinaryTreeInfluenceApproximation on an instance of the independent cascade model with an underlying binary modified arbitrary tree, T, with bin size ε is $O\big(\frac{|V(T)|}{\varepsilon}^6\big)$ with absolute error of at most $\varepsilon \cdot 2|V(T)|$.*

Proof. Following exactly the logic of Theorem 3.4.5, we see that we need to compute some number of table entries to generate our solution. The number of table entries is determined by the size of each of the bins. K can be at most $|V(T)|$ and there can be at most $\frac{2|V(T)|}{\varepsilon}$ possible values for $\sigma_{T_r}(S)$ and $\sigma_{T_r}(S \cup \{r\})$, because we can effectively double the number of nodes, so there are at most $|V(T)| \cdot \big(\frac{2|V(T)|}{\varepsilon}\big)^2$ table entries, yielding a worst case running time of $\mathrm{O}\big(\frac{|V(T)|}{\varepsilon}^6\big)$ following Theorem 3.4.7.

The absolute error is computed by multiplying the size of the bins times the number of nodes in the tree. Since we have twice as many nodes as before, the absolute error is at most $\varepsilon \cdot 2|V(T)|$. ■

Chapter 4

Conclusions

In this thesis we have shown how it is possible to determine maximum expected influence on at least one family of graphs using a specific diffusion model. By restricting the problem to a specific set of conditions, we were able to produce a method for solving an NP-hard problem. Although we were not able to provide a polynomial time exact solution for trees, we provided a strong polynomial time approximation result, which is stronger than the approximation of previous papers. By utilizing divide and conquer through dynamic programming, we have provided a base of knowledge for solving the expectation maximization problem on families of graphs.

We decided to apply the independent cascade model because its independence properties lend themselves to working with expected values and probability. This model has several properties that lend themselves well to the techniques we have used, but it may be constructive to determine whether our results hold for other commonly used diffusion models, such as the linear threshold model. One possible avenue of study is exploring how different models react to different families of graphs.

From our research we can infer that moving forward with other families of graphs, such as cycles, lattices, and graphs of fixed clique size or fixed treewidth, will require a much greater variety of programming and algorithmic techniques. The basic families of graphs we studied lent themselves well to the dynamic programming technique through their recursive, but such is probably not the case in other families of graphs. However, divide and conquer seems to be a natural way to examine an arbitrary graph family that has a recursive structure. It was frequently the case that we could take parts off of the graph, solve those parts, and then recombine the whole.

It is often the case, though, that while time complexity is reduced by dynamic programming, space complexity goes up. In our algorithms for paths and trees, we saw that we needed to determine the influence for each subpath with each relevant value of k. While space constraints were not necessarily relevant to our work, these constraints should be noted for any research that simulates our algorithms.

There is a wide range of future research that can be performed following our work. There are still numerous families of graphs on which expectation maximization can be attempted or which can have useful approximation results. It would be valuable to either find a polynomial time exact solution for trees or to prove that there is no such polynomial time algorithm, (assuming $P \neq NP$). It would also be valuable to explore other diffusion models in relation to families of graphs and see if there is a difference in complexity depending on what model is used. Lastly, it would be beneficial to solve the expectation maximization problem on families of graphs that are more informed by real-life problems or even find similarities in data sets and attempt to maximize influence on those data sets. We did not run any simulations of our algorithms and so were unable to compare our algorithms to those of others. Such simulations may be valuable. As this is a relatively young topic, we believe there are many avenues for further research.

Bibliography

[1] HL Bodlaender. A tourist guide through treewidth. *Developments in Theoretical Computer*, 1994.

[2] Gerard Cornuejols, Marshall L. Fisher, and George L. Nemhauser. Exceptional Paper–Location of Bank Accounts to Optimize Float: An Analytic Study of Exact and Approximate Algorithms. *MANAGEMENT SCIENCE*, 23(8):789–810, 1977.

[3] Pedro Domingos and Matt Richardson. Mining the network value of customers. *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '01*, pages 57–66, 2001.

[4] E Even-Dar and Asaf Shapira. A note on maximizing the spread of influence in social networks. *of the 3rd international conference on*, pages 281–286, 2007.

[5] Stephen Foster and Walt Potter. A history sensitive cascade model in diffusion networks. *Simulation*.

[6] Sharad Goel and Matthew J Salganik. Assessing respondent-driven sampling. *Proceedings of the National Academy of Sciences of the United States of America*, 107(1515):6743–7, 2010.

[7] Sharad Goel and M.J. Salganik. Respondent-driven sampling as markov chain monte carlo. *Statistics in medicine*, 28(1717):2202–2229, 2009.

[8] Jacob Goldenberg, B Libai, and E Muller. Talk of the network: A complex systems look at the underlying process of word-of-mouth. *Marketing Letters*, pages 211–223, 2001.

[9] Jacob Goldenberg, B Libai, and E Muller. Using complex systems analysis to advance marketing theory development: Modeling heterogeneity effects on new product growth through stochastic cellular. *Academy of Marketing Science Review*, 2001(9), 2001.

[10] Mark Granovetter. Threshold models of collective behavior. *American Journal of Sociology*, 83(6):1420, 1978.

[11] D.D. Heckathorn. Respondent-driven sampling ii: deriving valid population estimates from chain-referral samples of hidden populations. *Social Problems*, 49(1):11–34, 2002.

[12] D.D. Heckathorn. Extensions of respondent-driven sampling: analyzing continuous variables and controlling for differential recruitment. *Sociological Methodology*, 37(1):151–207, 2007.

[13] Douglas D. Heckathorn. Respondent-driven sampling: A new approach to the study of hidden populations. *Social Problems*, 44(2):174–199, 1997.

[14] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '03*, page 137, 2003.

[15] David Kempe, Jon Kleinberg, and Éva Tardos. Influential nodes in a diffusion model for social networks. In *IN ICALP*, pages 1127–1138. Springer Verlag, 2005.

[16] Elchanan Mossel and Sebastien Roch. On the submodularity of influence in social networks. *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing - STOC '07*, page 128, 2007.

[17] G.L. Nemhauser, L.A. Wolsey, and M.L. Fisher. An analysis of approximations for maximizing submodular set functions-i. *Mathematical Programming*, 14(1):265–294, 1978.

[18] Thomas Schelling. *Micromotives and Macrobehavior*. Norton, 1978.

[19] Erik Volz and Douglas D Heckathorn. Probability based estimation theory for respondent driven sampling. *New York*, 24(1):79–97, 2008.

[20] D. Wagner. Decomposition of partial orders. *Order*, 6(4):335–350, 1990.

[21] C Wejnert. An empirical test of respondent-driven sampling: Point estimates, variance, degree measures, and out-of-equilibrium data. *Sociological Methodology*, 2009.

[22] C. Wejnert and D.D. Heckathorn. Web-based network sampling: efficiency and efficacy of respondent-driven sampling for online research. *Sociological Methods & Research*, 37(1):105, 2008.

[23] Douglas B. West. *Introduction to Graph Theory*. Prentice Hall, 2nd edition, 2000.