Overexposure in Viral Marketing

May 27, 2020

1 Introduction

Our research addresses a recently raised computational question on information cascades (also known as diffusion) in social networks. The question is very general, but can be best illustrated using viral marketing as an example. With the advent of online social network platforms like Facebook and Twitter, marketing campaigns have shifted from primarily emails to more creative social messaging. However, trying to reach everyone indiscriminately comes with the risk of reaching those who view the product/message negatively. These critical individuals may hurt the information cascade by influencing their friends or by posting negative product reviews. This phenomenon of reaching unintended recipients during a cascade is called *overexposure*. Given a model of cascades with overexposure, the central computational question is: How do we select a set of "initial adopters" that will maximize the spread of a cascade? We plan to study this question in the budgeted setting, where we are allowed to select up to a certain number of initial adopters. We also plan to study the problem on a richer model of social networks compared to the previously studied models.

2 Previous Works

Many different models have been proposed for how to choose the optimal seed set for a given product. All of the work presented stems from the emergence of "influence maximization" (IM), an algorithmic problem in which the developer of a product assumes that a peer-influence process determines the product's adoption through a social network [**?**]. With limited resources, the problem is knowing which people to expose, or *seed* the product to first. Specifically, seeding is a marketing strategy where a new product is given away to a small portion of consumers (the "seeds") before launching the product to the entire market [**?**]. Many models used for seeding select seeds randomly, but other common strategies include choosing early adopters (those likely to purchase the product) or those with deep social connections (Social hubs) [**?**]. Within these seeding schemes, the risk of reaching clusters of negatively-inclined consumers who spread negative word of mouth (WOM) is still a new area of research. When negative WOM is present, trying to reach the largest number of people in a network cascade may not be the most profitable.

A recent study by Cui. et al, published in 2018, considers negative WOM in which recipients of negative WOM will spread that information faster and to a greater degree than positive WOM [**?**]. The model used in this study compares seed sets based on early adopters, social hubs, and randomly chosen seeds [**?**]. An agent-based modeling and simulation (ABMS) is used to simulate the adoption/rejection process of consumers in a network; consumers make the adoption decision based on both internal and external factors, p and q, respectively [**?**]. The internal factors are the effects of WOM, and the adoption/rejection/ignoring of each consumer is measured at different time periods, t. Each agent's probability of adoption is determined by the influence of positive(negative) WOM from each agent's neighbors. The two measures of performance used were net present value (NPV) and market penetration [**?**]. In this set-up, an agent's state (adopt or non-adopt) can change between time periods based on WOM. A few key results were found:

1. Seeding early adopters results in the highest NPV and market penetration when negative WOM is present.

2. When mean probability of consumers spreading negative WOM increases, relative impact of seeding early adopters increases.

3. For a lower quality product, seeding early adopters is still more effective for improving NPV and MP.

4. When products are more specialized, performance of seeding decreases.

[?] The model used assigns each agent in the market different attributes based on normal distributions:

- The probability to spread negative WOM, d
- Propensity to adopt the new product, p
- The effect of WOM on the connections, q

Whereas the simulation model presented by Cui et. al [?] compares networks with different seeding strategies, levels of internal influence, and probability of spreading WOM, this model does not use real data. There are also many different types of seeding strategies that can be used in conjunction with one another, rather than independently.

Another study by Iyer and Adamic [?] introduces the concept of "churn" into the model of overexposure for the IM problem, extending their previous work. The main question they seek to answer is whether it is optimal to seed everybody in the absence of budgetary constraints. When monotonicity is present, clearly it is optimal to seed everybody. However, when there is negative payout for rejecting a product, it may not be optimal to seed everybody in the network. [?]. Whereas most models of overexposure focus on the adoption/rejection binary, Iyer and Adamic considered the product experiences of people after adoption in their previous paper, considering a different network structure than traditional IM models [?]. In their newer paper, [?], they show that overambitious seeding can be detrimental in traditional threshold models as well once "churn" is introduced. The overall model used considered seeding one cluster, two clusters, and all three clusters in two models of social-product usage.

- The first model is the **repeated usage model**, where in a sequence of time steps, t, a person i may have access to a product. If they do, then they adopt the product with probability $p_i(t)$, with an initial probability of p_0 at $t = 0$. Then, each person has a threshold, s_i, which is the number of friends of i who must adopt the product for i to be satisfied. If s_i is met, then p_i increases by δ and decreases if not.
- The second model is called the **threshold model with churn**, where people are seeded into the adopting state, and others adopt if sufficiently many of their friends adopt. People can also transition into the non-adopting state from the adopting state, called **churning**. The state of each person is updated at each subsequent time step, t. Each person has accepting probability p_a.

To test these models, Iyer and Adamic ran simulations of the repeated usage model on Social Hash (SH) clusters of the Facebook friendship graph, selecting three cluster networks. For all values of p_0 except high values, universal seeding incurs costs. The proposed reasoning suggests that at early stages in the repeated-usage model, the rate of initial activity p_0 is low, so people seeded will adopt in unfavorable contexts. Thus, when p_0 is low, the costs of universal seeding are largest since activity is not sustained in the long term. Further, when p_0 is sufficiently high, a singly cluster can actually sustain long-term activity in isolation, making it costly to seed more than one cluster at $t = 0$ since the cluster can spread favorable contexts to the other clusters such that they also adopt.

For the threshold model with churn, universal seeding is beneficial at low values of p_a, where more social support is provided between users. As p_a increases, it becomes beneficial to seed two clusters and then finally only one when p_a is sufficiently high.

These two models were also tested with k-seeding, where the k-core of the network is seeded (subnetwork consisting of all nodes with at least k friendship edges). Specifically, 10-core seeding was tested, leading to more long-term adoption than universal seeding over the low p_0 regime for the repeated usage model [?]. For the threshold model, 10-core seeding wins out for higher values of p_a, similar to the cluster test [?].

Iyer and Ademic go on to analyze how it is the context of each individual's friends in the network that determines why overambitious seeding can be costly, such that when seeding results in contextually-unaware adoption choices, costs may be incurred. Further, overambitious seeding is more costly in the repeated-usage model than the threshold model; since p_a in the threshold model influences a person's willingness to adopt and the social support they get. When p_a is low, they will probably not adopt in the first place. In the repeated-usage model, p_0 determines how much social support someone gets and determines the time the person has an initial product experience. However, it does not determine whether the person uses the product in the first place, so at low p_0 many people can have bad experience and churn. Overall, Iyer and Ademic conclude that overambitious seeding can be costly when it results in "premature exhaustion of opportunities for further spreading" that would be more beneficial later in the spreading process [?].

Model Weaknesses:

In the present work, we can look to improve upon some of the weaknesses in other models which consider overexposure, primarily [?], [?], [?]. Starting with Cui et al. [?], one of the biggest weaknesses is that it uses agent-based modeling and simulation (ABMS) rather than real data. This model takes into account the non-linear affect of negative WOM when simulating the diffusion process over time. An algorithm is describes which generates N consumers with a propensity to adopt, q, and then adds ties between consumers at each successive time step t. A random distribution determines the attributes of each consumer (probability to spread negative WOM, propensity to adopt, and effect of WOM on connections). A seeding strategy is chosen and then tested–either social hubs, early adopters, or random. Rather than picking from a normal distribution, it would be beneficial to test the model using real-world networks, such as the Facebook clusters studied in [?] and the email networks in [?]. In addition, building off the simulation, the model could be improved upon by allowing more than one of the three seeding strategies to be used at the same time, for instance, seeding both early adopters and social hubs. The last weakness we will touch upon for our purposes is that once a consumer is in one of the three states (satisfied adopter, disatissfied adopter, or rejecter), they will not change state [?]. It would be interesting to allow consumers to change state such as in [?], such that changes in satisfaction level by both internal and external factors are considered.

Moving to the model presented by Iyer and Adamic [?], there are weaknesses as a result of a lack of clarity in how it is defined and a lack of generality. First, the narrative describing the model is a bit confusing–it is not well-defined how the simulation is being updated at each time interval. Is the probability of each individual using the product changing from t to $t+1$ only as a result of s_i being met and δ being added/subtracted to p_i? Or, can p_i change as a result of internal factors, where the individual's own experience makes them more or less likely to continue using the product? Further, the model at hand only considers s_i values that are constant for everyone in the network, rather than taking on a range of values within the network. Looking at simulations of the repeated usage model and threshold model where s_i takes on a range of values would be interesting to look into. The model could also have smoother calculations by allowing s_i to be a percentage, where rather than a threshold number of friends, each consumer has a threshold percentage of accepting neighbors. Another large weakness of this paper is that the example used is very specific, where the 3 SH clusters are seeded in different amounts. This network is very specific and the seeding strategy can really only be applied when there are clusters as well-defined as Facebook SH clusters. Last, the model does not consider budgetary constraints; for instance, the company/firm may not be able to afford seeding all three clusters, and the problem would then shift to maximizing profit given a constraint on the cost of seeding. The k-core model proposed seems similar to the structure of the model in [?], where there is a budget k for the size of the seed-set. Since the budgeted problem in [?] is NP-complete, it might be beneficial to look at the k-core as our "budgeted" case, thereby slightly changing the problem stated in [?].

Going off of this, the largest weakness for our purposes in [?] is that it follows a binary, where individuals are either adopting or not adopting, and do not ever change their state. Rather than looking at time intervals, this model simply looks at the fixed threshold θ of each individual in the network. The effects of negative WOM are not considered since θ does not change based on whether or not someone's neighbor adopts the product. In this way, the model in [?] is simpler than other models we have looked at, and it would be beneficial to try and add more flexible paramters to the cascade model proposed.

3 Key Differences

The model that we seek to improve upon in this paper is the one presented by Abebe et al. [?]. Specifically, we are looking at improving approximation algorithms for the budgeted seeding case, which was proven to be NP-complete. However, there are key differences between the model presented in [?] and those of [?] and [?]. Specifically, we have:

- **How overexposure happens:** In [?], they are including a negative payoff for rejection, where the firm will lose money if someone does not adopt. This is how overexposure can happen, since many rejecters can lead to negative payoff. However, Ayer and Ademic. [?] do not have a negative payoff, but rather overexposure happens by a threshold model, where adoption/not adoption is impacted by how many neighbors are adopting
- **Probability vs. Adoption Threshold** In [?], there is a fixed appeal to a product, ϕ, and each node is associated with a fixed criticality threshold, θ, such that if $\theta < \phi$, the node adopts. However, in [?], each person adopts via a certain probability, p, which can change based on how many neighbors are adopting.
- **Time Steps** In [?], we can look at the payoff function for the entire graph in one instance since all θ_i are known and do not change. However, in [?], the model used proceeds in a series of time steps, where only the seeds are exposed at time $t = 0$ and adopt with probability p_0. Then, their immediate neighbors get access at $t = 1$, but their probability of adoption p_1 is changed based on how many neighbors of theirs are currently using the product. Thus, someone who may not adopt at earlier stages may adopt at a later stage if enough of their neighbors are.
- **Propagation:** The cascade model in [?] is structured such that one a rejecting node is reached, the message is not sent to their neighbors. In [?], the message continues to propagate out.

Our goal is to improve the budgeted case based on the current model in [?] as well as include more parameters into the model based on [?] and [?]. For instance, we hope to include some sort of threshold that increases or decreases each agent's criticality parameter based on how many of their neighbors are adopting. In addition, we can make larger changes to the model by introducing time steps into the model, where the initial seeds can be chosen based on the k-core model proposed in [?] or approaches in [?] where the agents with the highest p_a are seeded first.

4 Groupon Effect

One key example of the negative effects of overexposure is given by Adamic et al. in [?], where viral marketing via Groupon coupons leads to lower Yelp ratings. The proposed hypothesis by [?] in their new paper is that Groupon reviews are by real customers, whereas the baseline of other reviews is artificially high due to businesses generating "fake" positive reviews. The study marks discontinuities in average rating scores of two different data sets on Yelp, such as lower overall reviews from Groupon users and a decline in reviews close to the Groupon expiration date. The team came up with a few key hypotheses, backed by data analyses. Some of these include: [?]

- Intrinsic Decline over time, independent of Groupon
- Groupon users are more critical reviewers
- Businesses using Groupon are desperate, in trouble anyway
- Businesses discriminate against customers with a groupon, giving poorer service
- Artificial reviews are present in the reviews, so non-Groupon reviews are inflated

In this way, the Groupon effect may be independent of overexposure in certain cases, or compounded by other factors. As with any review analysis, finding empirical data is difficult due to the presence of fake reviews, a lack of reviewers, and varying degrees of usefulness. In other words, a decline in reviews does

not necessarily correlate to a decline in business, and a slight decline in reviews may indicate more "real" people using the product and thus an increase in profit. The effects of overexposure must be analyzed in more scenarios than Yelp reviews; further, the Groupon effect in itself needs more empirical data to have a solid case.

5 Edge Covering Problems

The overarching goal of this project is to find a way to approximate the budgeted case of the problem of overexposure, where the firm can seed at most k nodes so as to maximize overall profit. Since the payoff function with overexposure is neither submodular or supermodular, it is NP-hard to decide if there is a budgeted set yielding positive payoff in the first place [**?**]. This problem is similar to the k-edge-incident subgraph problem discussed in [], which is to find a set of nodes $W \subset V$ of maximum weight such that the number of edges with at least one endpoint in W is at most k. It is proved in [] that since the MAXCLIQUE problem is NP-complete, the k-edge-incident subgraph problem must be NP complete. However, there are greedy algorithms presented to approximate the problem in linear time. For the weighted case, which is what we are working with, the greedy algorithm is a special case of the Knapsack algorithm, solved by dynamic programming in $O(k|V|)$ time. Our model can seek to incorporate these greedy algorithms, since profit-maximization in the budgeted case seems very similar to the k-edge-incident subgraph problem given the graph set-up in [**?**].

6 Model Set-Up

The present work builds off the models presented in [**?**] and [**?**]; specifically, we will be looking at the effects of overexposure from two angles: there are costs when a customer rejects the product, and the customer then spreads negative WOM, making others less likely to adopt the product. We also focus on the *budgeted* case, where the firm can seed up to k nodes. Since the budgeted case was proven to be NP-hard for the model proposed in [**?**], we seek to find approximation algorithms for our own model to optimize payoff with k seeds.

Preliminaries

We will work off the Generalized Model from Abebe et al, [**?**]. In this model, a product has a paramter $\phi \in [0, 1]$, measuring the breadth of its appeal. Each consumer, i in the network has a criticality paramter θ_i, such that they adopt the product if $\theta_i < \phi$ and rejects it otherwise. Then, we also have parameters σ_i and τ_i that determine whether the consumer spreads WOM. In our model, if a consumer decides to spread WOM, it not only means that their neighbors have access to the product, but also that the threshold of their neighbors' goes up or down. Thus, we have that each agent starts as one of 4 types:

1. **Rejecter:** Agents for which $\phi < \tau_i$. These users reject the product and do not spread negative WOM to their neighbors. There is no negative payoff since the users do not actually accept the product.

2. **Dissatisfied Adopter:** These users have $\phi \in [\tau_i, \theta_i]$, and accept the product but do not like it, and therefore have a negative payout of $-q < 0$. They spread negative WOM to their neighbors.

3. **Neutral Adopters:** These users have $\phi \in [\theta_i, \sigma_i]$. They accept the product, for a payoff of $p > 0$, but do not spread WOM.

4. **Satisfied Adopters:** These users have $\sigma \geq \sigma_i$, such that they accept the product and spread positive WOM to their neighbors.

In our model, spreading WOM can be encapsulated by the parameter $\delta > 0$ such that neighbors of consumers who spread positive(negative) WOM will have lower(higher) thresholds as a result. Then, for a given node, we let C_i^+ be the total number of neighbors giving positive WOM and C_i^- be the total number of neighbors giving negative WOM. Thus, we have, for each agent i:

$$\theta_i' = \theta_i + (p \times C_i^+) + (n \times C_i^-)$$

where p, n represent the magnitude of positive, negative WOM for consumers in the network. In this way, having lots of neighbors that are dissatisfied/reject a product will make it more likely for a consumer to also reject the product, even if their initial θ was higher. We will include time steps in our model as well, updating the threshold of nodes in the network at each time step. We will follow Abebe et al. and say that a cluster is determined by doing BFS from a given node, adding nodes to the cluster if they adopt the product and stopping each iteration down a path when a node for which $\theta > \phi$ is reached.

In order to transition from the model in Abebe et al. to a linear threshold model as in Ayer and Adamic, we first need to generalize the threshold model used in Abebe et al.

7 Optimization Algorithms

For the following approximation algorithms, we have:

- $n \in \mathbb{N}$ is the number of nodes in the graph
- k is the number of nodes we are choosing to seed

1. Pseudo-Greedy Algorithm

This first approach is a pseudo-greedy algorithm such that it adds nodes to the seed set one at a time, optimizing profit for each addition. Essentially, the first iteration of the outer for-loop picks the single node with the highest payoff. Then, that node is removed from the seed set. On the second iteration, another node is picked which maximizes profit for two seeds. All of the nodes that were not picked from the first iteration are considered. The algorithm continues this way for k iterations, returning the set of nodes that optimizes payoff.

Algorithm 1 Pseudo-Greedy Algorithm

```
1:  procedure GREEDY(G, seedSet, phi, k)
2:      kSet = []
3:      payoff = -2
4:      for seed in (1, k + 1) do                          ▷ number of seeds
5:          Do this
6:          for node in seedSet do
7:              kSet.append(node)
8:              payoff2 = computePayoff(G, kSet, phi)
9:              if payoff2 ≤ payoff: then
10:                 kSet.pop()
11:             else
12:                 payoff=payoff2
13:                 kSet.pop()
14:             kSet.append(node)          ▷ add the node that maximizes payoff to the kSet
15:             seedSet.remove(node)       ▷ Remove so we do not consider the same node again
16:             if payoff > 0 : then
17:                 return kSet
18:             else
19:                 return []
```

2. Dynamic Programming Algorithm The next approach is a Dynamic Programming Algorithm using a table-based bottom-up approach. Rather than looking at individual nodes, this algorithm looks at clusters of accepting nodes separated by "walls" of rejecting nodes.
Essentially, each node in the graph is a cluster of accepting nodes, and each edge between two cluster nodes, say $m_{1,1}$ and $m_{2,2}$, represents all of the rejecting nodes that are adjacent to a node in $m_{1,1}$ and a node in $m_{2,2}$.

Algorithm 2 Dynamic Programming Algorithm

```
procedure DP(G, n, k)
    storePayoff = [i][k]
    storeSeeds = [i][k]
    for seed in (0, k) do                                   ▷ number of seeds
        Do this
        for node in (0, n) do
            if node == 0 and seed == 0: then
                storeSeeds[seed][node] = [node]             ▷ Store node numer
                nodePayoff = computeNegPayoff(G, node)      ▷ calls method that subtracts edge weight
                storePayoff[seed][node]=nodePayoff
            else
                last = storePayoff[seed-1][node-1]
                nextGuess = computeNegPayoff(G, node) + last
                for lastNodes in storeSeeds[seed-1][node-1] do
                    neighbors = nx.neighbors(G, lastNodes)
                    for nodes in neighbors do
                        if nodes == node: then              ▷ ensures no double-counting of edges
                            add = weight of edge (node, lastNodes)
                            nextGuess += add
                lastEntry = storePayoff[seed][node-1]
                storePayoff[seed][node] = max(lastEntry, nextGuess, last)
                storeSeeds[seed][node] = seeds from best choice above
    return (storePayoff[k-1][i-1], storeSeeds[k-1][i-1])
```

The graphs below represent the graph with each node representing one agent (right) and then each node representing a cluster of accepting agents, with each edge representing a wall of rejecting agents (red in the right graph)

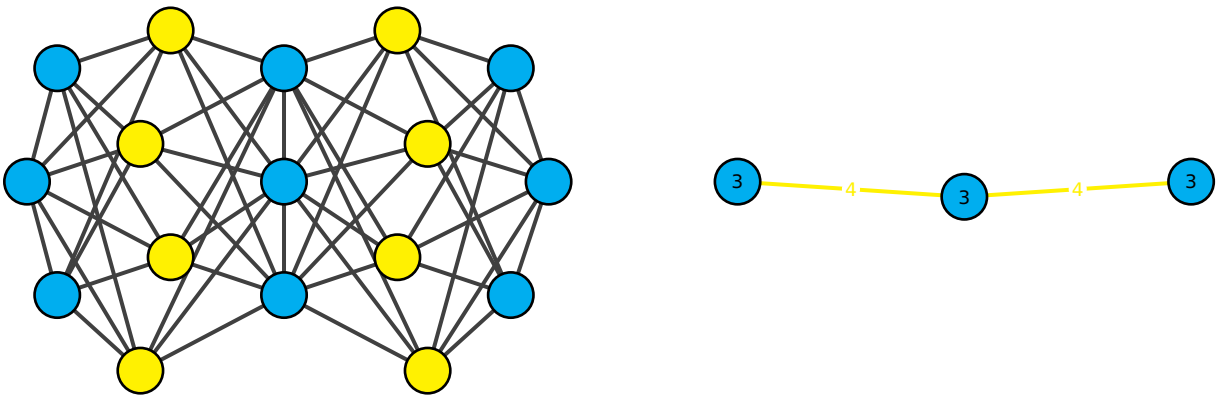

Figure 1: Individual Nodes vs Cluster Nodes

[H]

So, when we run the dynamic programming algorithm, we are picking k clusters, seeding one agent in each cluster. This works since we know that every agent in a cluster is accepting, so all the agents in the cluster will accept the product eventually if we seed one individual. Thus, we run dynamic programming on the weights of each cluster node.

8 3. Linear Program

The next approach was to design a simple linear program with linear objective function and linear constraints. We have the following set-up:

- n is the number of nodes in the graph, and $k \leq n$ is the number of nodes we are choosing to seed
- x is a matrix of variables for each of the possible edges in the graph. These are the variables for the linear program. The variables are each either 1 (take) or 0 (don't take):
  - $x = \text{Matrix}[m_{i,j}\ \forall i,j \leq n] \implies x$ is an $n \times n$ matrix
  - Then, this gives us:

$$x = \begin{bmatrix} m_{1,1} & m_{1,2} \cdots & m_{1,n} & \\ m_{2,1} & m_{2,2} & \cdots & m_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n,1} & m_{n,2} & \cdots & m_{n,n} \end{bmatrix}$$

  - Thus, each diagonal entry corresponds to a particular node, so $m_{i,i}$ is a node and $m_{i,j}$ is an edge connecting $m_{i,i}$ and another node $m_{j,j}$, for $i,j \leq n$. We either take a node (and its corresponding edges) or we don't. All of the entries of x take a value of 1 or 0 – if we take a node (diagonal entry) its value is 1 and then all of its corresponding edges will also be 1. If we don't take a node, its value is 0 and we set all of its edges to 0.
- w is a matrix of weights for each node and edge combination, and is also $n \times n$. Each entry $w_{i,i}$ is the weight of the node $m_{i,i}$, while each entry $w_{i,j}$ is the weight of the edge $m_{i,j}$
- Then, the following matrix has $w_{i,j} \in \{0\} \cup \mathbb{N}$, $\forall i,j \leq n$

$$w = \begin{bmatrix} w_{1,1} & w_{1,2} \cdots & w_{1,n} & \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n,1} & w_{n,2} & \cdots & w_{n,n} \end{bmatrix}$$

  - In other words, w is a matrix of integer weights for the nodes/edges in x

We now have the following objective function, where n is the number of nodes in the graph:

$$\text{objective} = \sum_{i=1}^{n} m_{i,i} * w_{i,i} - \sum_{e=1}^{n}\sum_{j=1}^{n} m_{e,j} * w_{e,j}$$

This objective function adds up the payoff from each cluster, then subtracts the sum of the negative payoff associated with each cluster. The linear program considers shared walls of rejecting nodes so that if two clusters share a wall of rejecting nodes, the negative payoff is only subtracted once. Remember that in our case, $m_{i,i}$ is on the main diagonal, corresponding to a node. In contrast, $m_{i,j}$ (non-diagonal) is an edge. The nodes of the graph are accepting clusters of nodes, while the edges are rejecting nodes between clusters. The weight of each node is the number of accepting agents in the cluster, and the weight of each edge is the number of rejecting agents two clusters share.

The following constraints are applied to the objective to ensure that if a node is chosen, so are its adjacent edges, making sure that no edge is double-counted:

- $1 \geq m_{i,j} \geq 0$, such that $m_{i,j} \in \mathbb{Z}$ for all $i,j \leq n \implies$ we either take a node and its corresponding edges (1), or we don't (0).
- If $w_{i,j} > 0$, then $m_{i,j} \geq m_{i,i}$ for all $i,j \leq n$ and $i \neq j \implies$ if the edge exists (meaning its weight isn't zero), then we must consider that edge. If the node $m_{i,i}$ is chosen, then $m_{i,i} = 1$, and this constraint ensures that all adjacent edges $m_{i,j}$ will also be 1. its corresponding edge will be considered. If the node $m_{i,i} = 0$, then it will be optimal to let $m_{i,j} = 0$ since we are subtracting the weights of the edges

- $\sum_{i=1}^{n} m_{i,i} \leq k$, where k is the number of nodes we are choosing to seed.

Running this linear program with the entries of the matrix x as variables and the corresponding matrix of integer weights, subject to the constraints above, will give us an optimal seed set. The program will return the matrix x, where each diagonal entry with a value of 1 is a node which is optimal to seed.

9 Comparing DP to LP

Num Nodes	Num Seeds	LP Payoff	LP SeedSet	LP Time
20	8	56	[3,5,7,12,13,17,18,19]	0.27
100	8	95	[84,83,55,52,30,36,21,38]	1.19
200	8	101	[29,79,82,83,100,141,159,197]	12.31
300	8	97	[11,90,112,169,184,198,232]	44.84
300	20	223	see below	55.39
500	20	225	see below	466.22
500	40	466	see below	453.03

[6,21,26,60,62,66,78,102,127,128,156,162,176,184,198,234,264,294,298]

142,149,181,166,198, 221,291,316,324,362,376,383,391,427,462,474,495

Num Nodes	Num Seeds	DP Payoff	DP SeedSet	DP Time
20	8	53	[5,7,10,12,13,17,18,19]	0.00501
100	8	95	[21,30,36,38,52,55,83,84]	0.03109
200	8	98	[0, 40, 69, 101, 142, 159, 183, 184]	0.06278
300	8	97	[11, 86, 90, 99, 169, 184, 198, 246]	0.1401
300	20	217	see below	0.49
500	20	220	see below	1.77552
500	40	465	see below	4.154890060424805

[6,26,34,54,60,66,78,102,127,128,162,176,192,194,198,234,244,264,294,298]

36, 54, 141, 142, 149, 166, 181, 198, 221, 291, 316, 324, 362, 376, 383, 391, 427, 462, 474

11, 13, 27, 30, 40, 43, 99, 101, 107, 108, 131, 140, 162, 172, 176, 202, 222, 242, 249, 263, 264, 273, 279, 281, 311, 334, 361, 374, 383, 388, 396, 411, 419, 425, 446, 459, 475, 480, 495

With larger payoff values:

Num Nodes	Num Seeds	LP Payoff	LP SeedSet	LP Time
200	30	2730000	0.27	

Num Nodes	Num Seeds	DP Payoff	DP Time
200	30	2670000	11.358106136322021

Main takeaways from the Comparison:

- Linear programming in mathematica is much slower than Python DP, likely due to the large matrices used in the representation. In the DP algorithm, the graph data may be easier to access than the matrix values. Also, Mathematica uses exact line search

10 Improved DP

We can improve Dynamic Programming for trees by partitioning the seeds among the subtrees based off the children of the root node. We then recursively do this for the children at each level of the tree, partitioning

the seeds and making recursive calls on the children and that number of seeds. Say, for instance, we have a tree with 15 nodes, and are seeding 4 seeds. Then, if the root node has 2 children, our recursive calls from the root will be all the partitions of 4 into 2, namely:

- $[0, 4]$
- $[4, 0]$
- $[2, 2]$
- $[3, 1]$
- $[1, 3]$

Then, each of these will be a recursive call on the left and right subtrees, respectively. Note that we also have to consider the problem of the root, namely, that we are including the root in both the left and right subtrees, which could be a problem as we are looking at the subtrees independently, but they both share the root. For instance, say the left subtree chooses the root; then this decision has to be made while considering the edge between the root and the right subtree. One solution is to also recurse TWICE on each subtree, once where we seed the root and once where we don't. Otherwise, there would be no easy way to track whether or not we are choosing to seed the root. Of course, we will quickly get overlapping subproblems so this should not be a huge pain.

The other problem now is how we keep track of the decisions made as we propagate back up the recursive call chain. Recursion should handle this, as we are recursing so that at the root we are partitioning the k seeds. The base cases would be as follows:

- we have zero seeds, $k = 0$
- we are at a leaf node, $children = Null$

We need to pass the following parameters into the recursive call:

- the subtree we are recursing on
- the number of seeds
- root or no root
- the source node (parent of subtree)

What does root or no root mean?

In the example, we will recurse on the right and left subtrees, considering and not considering the root. So, we are either forcing the algorithm to choose the root or forcing the algorithm to not choose the root. This gives us:

- $R(0, noroot), L(4, noroot)$
- $R(4, noroot), L(0, noroot)$
- $R(2, noroot), L(2, noroot)$
- $R(3, noroot), L(1, noroot)$
- $R(1, noroot), L(3, noroot)$
- $R(0, root), L(3, root)$
- $R(3, root), L(0, root)$
- $R(1, root), L(2, root)$

- $R(2, root), L(1, root)$

The memoization table will consist of the number of the node which is the root of the tree, the number of seeds in the partition for that tree, and root or no root. Thus, it will be a 3D table (as above, we see we could have R(2, root) and R(2, noroot) which will lead to potentially different outcomes so we must consider them as such).

10.1 Next problem: how do we ensure optimal substructure?

I think it's time for some pseudocode to think about this...

Algorithm 3 Dynamic Programming Algorithm

```
1: procedure DP(G, k, source, root)
2:     storePayoff = [source][k][root]                          ▷ memoization table
3:     if k == 0 :                                              ▷ meaning we have no seeds
4:         return 0
5:     if G.children = null                                     ▷ we are at a leaf node
6:         return 0
7:     if s thentorePayoff[source][k][root] != 0:
8:         return storePayoff[source][k][root]
9:     childen = nx.children(G, source)
10:    partitions = part(k, len(children))  ▷ call partitions function to partition the k seeds throughout the subtrees
11:    for p in partitions: do
12:        for i in range(numChildren): do
13:            G_sub = G.subgraph(i)                            ▷ make a subgraph
14:            payoff_root = computePayoff(source) + DP(G_sub, p[i] − 1, source, root = True)
15:            payoff_noRoot = DP(G_sub, p[i], source, root = False)
16:    return max(payoff_root, payoff_noRoot)
```

to seed liographystyleplain References