Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Master's Thesis Nr. 117

Systems Group, Department of Computer Science, ETH Zurich

Towards Revenue Maximization by Viral Marketing:
A Social Network Host's Perspective

by

Benjamin Zehnder

Supervised by

Dr. Arijit Khan
Prof. Dr. Donald Kossmann

15.03.2014 - 15.09.2014

inf | Informatik
Computer Science

Abstract

We study the novel problem of revenue maximization of a social network host that sells viral marketing campaigns to multiple competing campaigners. Each client campaigner informs the social network host about her target users in the network, as well as how much money she is willing to pay to the host if one of her target users adopts her product due to the influence of her campaign. Because of various product-adoption costs and preferences, it is very unlikely that an average user will adopt more than one of the competing products. Therefore, from the host's perspective, in order to maximize her own revenue, it is important not only to choose the seeds that maximize the collective expected spread of all campaigns, but also to assign seeds to client campaigners in such a way that it guarantees the maximum expected spread of each campaign inside the corresponding campaigner's target set.

We formulate our problem by following two well-established information diffusion models: the independent cascade model and the linear threshold model. While our problem using both these models is **NP**-hard, and neither monotonic nor submodular; we design efficient algorithms with theoretical performance guarantees. Our detailed empirical evaluation attests that the proposed techniques are effective and scalable over real-world datasets, and significantly outperforms state-of-the-art viral marketing techniques.

Contents

Chapter 1

Introduction

The classical viral marketing problem [1, 2] identifies the top-k seed users in a social network such that the seed users can maximize the spread of an information in the network. The budget k on the seed set size usually depends on the campaigner — in other words, it depends on how many initial users the campaigner can directly influence through free samples, discounted prices, direct advertisements, and so on. The bulk of the research in the domain of viral marketing assumes that the social network structure is available to the campaigners. However, in real-world scenarios, the social network platforms are owned by third-party hosts [3], such as Facebook, Twitter, LinkedIn, and LiveJournal; and the hosts keep their social graphs secret for their own benefits as well as due to privacy reasons. Therefore, marketing companies themselves are not able to run viral marketing campaigns due to lack of access to the social network graph.

Motivated by the above reasonings, in this study, we assume that the viral marketing campaigns are run by the social network host on behalf of its clients, who are the marketing campaigners. Each campaigner informs the host about: **(a)** her budget on the seed set size, and also **(b)** how much money she is willing to pay to the host for each of her target users if that user adopts her product due to the influence of her campaign. In the real-world, multiple companies compete over comparable products (e.g., Nintendo's Wii vs. Sony's Playstation vs. Microsoft's X-Box; Microsoft's Surface vs. Apple's iPad vs. Google's Android) [4, 5, 6, 3]; and therefore, the host needs to *simultaneously* run multiple competing viral marketing campaigns over the network. Due to various product-adoption costs and preferences, it is very unlikely that an average user will adopt more than one of the competing products. Since most of the users usually adopt only one of the competing products, it also implies that the seed sets of the competing campaigners require to be mutually non-overlapping [3]. Therefore, from the host's perspective, the challenge lies in the fact of how she selects the seed set for each of her client campaigners so that the host's overall expected revenue is maximized.

We must emphasize that our problem of viral marketing from the host's perspective is very similar to the idea introduced earlier in [3] by Lu et al.

Nevertheless, [3] does not study our problem of maximizing the host's revenue, rather they consider how to balance the expected spread of each campaign over the entire social network. In addition, [3] does not apply the notion of target users for each client campaigner; but in reality, a campaigner is often willing to pay more money to the host if her target users adopt her product. More specifically, for different users, each campaigner might be willing to pay a different amount of money to the host if those users adopt her product. Clearly, maximizing the host's overall expected revenue is the problem of interest for most practical scenarios.

A naïve approach to solve our revenue maximization problem would be to apply the classical viral marketing algorithms [2, 7] *separately* for each campaigner as follows. We first define an arbitrary order among the campaigners, and then identify the top-k seed nodes for each campaigner such that the host's revenue is maximized by considering one campaigner at a time. If some of the top-k seed nodes for the current campaigner have already been assigned to previous campaigners, we identify the next-best seed nodes for the current campaigner until we exhaust the budget k of the current campaigner's seed set size. However, the host's aggregated revenue by following this naïve approach could be sub-optimal, because in real-world scenarios, the host needs to run all the viral marketing campaigns *simultaneously* in the network. Therefore, if one identifies the seed nodes separately for each campaigner, the method would be unable to capture the competition among multiple campaigns running simultaneously in the network; and hence, the host's aggregated revenue could be sub-optimal. We illustrate the difficulties with the baseline approach with an example below.

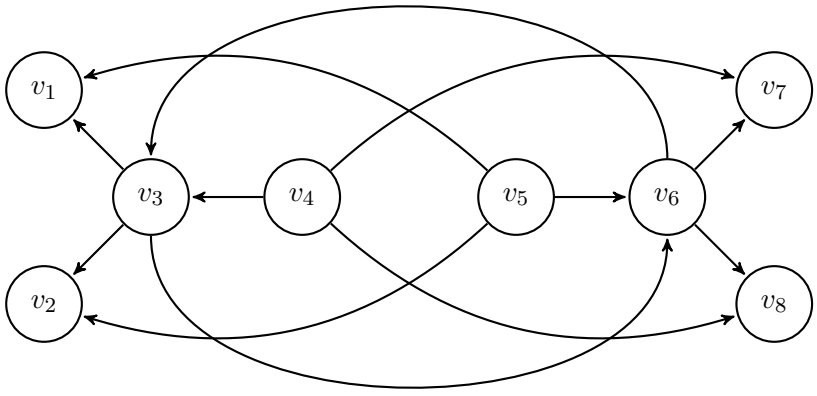


Figure 1.1: Revenue Maximization: Difficulties with Naïve Methods

Example 1 Let us assume that there are two campaigners — C_1 and C_2, who are viral marketing clients to the host of the social network depicted in Figure 1.1.

We also assume that each directed edge has an influence probability 1, and we apply the classical independent cascade model [2] for information diffusion. Let A_{ij} denote the money that campaigner C_i is willing to pay to the host if node v_j is influenced by her campaign. We assume $A_{11} = A_{12} = A_{13} = A_{14} = 10\$$, $A_{15} = A_{16} = A_{17} = A_{18} = 1\$$; while $A_{21} = A_{22} = A_{23} = A_{24} = 1\$$, and $A_{25} = A_{26} = A_{27} = A_{28} = 10\$$. We also assume that the budget on the seed set size for each campaigner is only 1.

Now, if we apply the aforementioned naïve method, that is, we identify the seed set for each campaigner separately by running a viral marketing algorithm, then we shall get v_4 as the best seed node for C_1. This is because v_4 could eventually influence v_1, v_2, v_3, v_6, v_7, and v_8, and the host will get a revenue of 43\$, assuming C_1 is the only campaigner in the network. Similarly, we shall find that v_5 is the best seed node for C_2, assuming there is no other campaigner. Now, in reality, when the host runs the two viral marketing campaigns simultaneously with the campaign of C_1 starting from v_4 and that of C_2 starting from v_5, the host's aggregate revenue will be only 44\$. This is because after simultaneous campaigning, v_3, v_4, v_7, and v_8 will be influenced by C_1, while the remaining nodes will be influenced by C_2.

A better solution from the host's perspective indeed exists. For example, if v_3 and v_6 are selected as the seed nodes for the campaigners C_1 and C_2, respectively, then v_1, v_2, v_3 will be influenced by C_1, while v_6, v_7, and v_8 will be influenced by C_2. This will ensure an aggregated revenue of 60\$ to the host. This example clearly illustrates that running a viral marketing algorithm separately to find the seed set for each campaigner is not the best option from the host's perspective, and it may result in a sub-optimal aggregated revenue for the host. □

Our Contributions and Roadmap. Our contributions can be summarized as follows:

- We define the fundamental and novel problem of host's revenue maximization by viral marketing in the presence of $m \geq 2$ competitive campaigners (Section 3.1).
- We formulate the problem using two widely-used information diffusion models — the independent cascade (IC) model (Section 3.5) and the linear threshold (LT) model (Section 3.6).
- We show that our problem using both these models is **NP**-hard and neither submodular nor monotonic; hence, we design effective and efficient heuristic schemes to solve our problem. For the LT model, we also provide a theoretical performance guarantee of $\frac{1}{m}(1 - \frac{1}{e})$ for our algorithm (Section 3.6). In addition, we show that our problem can be solved exactly in polynomial time over a tree dataset using both these models.

- We conduct a thorough experimental evaluation using several real-world datasets with various kinds of revenue distributions (Chapter 4). Our empirical results (Chapter 5) attest that our methods generate high-quality results as compared to baselines, especially when the campaigners are constrained by a small number of seed nodes, as well as over tree datasets.

Chapter 2

Related Research

Most of the content of this master's thesis is contained in our paper [8], because the paper was created during this thesis, nevertheless we present additional methods, proofs and results in this master's thesis.

Viral marketing aims at finding a set of seed nodes that generates the largest expected information cascade in a social network. Domingos and Richardson [1] formulated viral marketing as an optimization problem, while Kempe et al. [2] proposed the linear threshold model and the independent cascade model, as well as designed approximation algorithms with provable performance guarantees. Since then, several heuristics have been proposed to improve the efficiency of that method [9, 10, 11, 12, 13, 14, 15]. Very recently, [16, 17] developed almost linear-time viral marketing algorithms, yet providing the same approximation guarantee as Kempe et al.'s original method. In future work, it will be interesting to apply these efficient viral marketing techniques [16, 17] as the underlying method to improve the overall efficiency of our revenue maximization algorithms.

There are several works on viral marketing in the presence of a competing negative information cascade [18, 4, 5, 19, 6]. In [7], Lappas et al. introduced the concept of target marketing and k-effectors — by identifying k seed nodes such that the spread of an information is maximized over a set of given nodes and minimized outside that set. Both [20, 21] also considered the notion of target marketing by maximizing the influence over a region of the social network.

While the bulk of the research in the domain of viral marketing assumes that the social network structure is available to the campaigners; in reality, the social network platforms are owned by third-party hosts. Lu et al. [3] were the first one to consider the viral marketing problem from the social network host's perspective. In their framework, the host sells simultaneous viral marketing campaigns, as well as it selects the seed nodes on behalf of its client campaigners such that the "bang for the buck" for each client company is nearly the same. Nevertheless, [3] did not consider our problem of maximizing the host's revenue. Moreover, [3] did not consider the notion of target users for each client campaigner; but in reality, a campaigner is often willing to pay more if her target users adopt her

product; and also each campaigner might pay a different amount of money to the host even for the same user when that user adopts her product. Clearly, this is the problem of interest for most practical scenarios.

Chapter 3

Theory

In this chapter we introduce the problem statement and two different information diffusion models, which we use for the information propagation through the graph. We then proof that our problem is **NP**-hard and neither monotonic nor submodular under both information diffusion models. Next we present a simple baseline solution and our solution for each information diffusion model. At the end of this chapter we show some of the possible extensions to our problem statement and the corresponding adjustments to our solutions.

3.1 Problem Statement

A social network $\mathcal{G}$ is modeled as a triple (V, E, P), where V is a set of n nodes, $E \subseteq V \times V$ is a set of e directed edges, and $P : E \to (0, 1)$ is a probability function that assigns a probability to each edge in E. More specifically, the probability p_{uv} on a directed edge $(u, v) \in E$ represents the information diffusion probability along that edge. Note that this diffusion probability simply indicates the probability that an exposure of node u to some information also results in that information being assimilated by node v. Node v then automatically becomes eligible to transmit that information to its neighbors. We shall discuss the details of various information diffusion models in Section 3.2.

We consider $m \geq 2$ competing campaigners, denoted by $C_1, C_2, \ldots, C_m$, for whom the social network host runs simultaneous viral marketing campaigns. Each campaigner C_i specifies a budget k_i as his seed set size. We denote by S_i the seed set for campaigner C_i, and A_{iu} the money that campaigner C_i is willing to pay to the host if node u is influenced by C_i's campaign. We refer to $A = (A_{iu})_{m \times n}$ the *revenue matrix*. We also denote by $Pr(u, i, S)$ the probability that node u will be influenced by C_i's campaign following some information diffusion model, where $S = \{S_1, S_2, \ldots, S_m\}$ represents the collection of the seed sets for the m campaigners. We are now ready to define our problem.

Problem 1 (Revenue Maximization) Given a network $\mathcal{G} = (V, E, P)$, $m \geq 2$ client campaigners, the revenue matrix A, and a budget k_i on the seed set size for each campaigner C_i, i.e., $|S_i| = k_i$ for $1 \leq i \leq m$, find the seed set for each campaigner such that the expected revenue of the host is maximized. Formally,

$$\begin{aligned} &\arg\max_{S_1,S_2,\ldots,S_m} \sum_{i=1}^{m} \sum_{u \in V} [A_{iu} \cdot Pr(u, i, S)] \\ &\text{such that} \quad |S_i| = k_i \quad \forall i \in (1, m) \\ &\text{and} \quad S_i \bigcap S_j = \phi \quad \forall i \neq j; \quad i, j \in (1, m) \end{aligned} \tag{3.1}$$

3.2 Information Diffusion Models

In this section we present the two information diffusion models we use in this master's thesis. Both, the independent cascade (IC) and the linear threshold (LT) propagation model, are widely-used for problems with one campaigner [2]. For multiple campaigners both propagation models need to be adjusted. In the following subsections the original models and the adjusted models are described in detail.

3.2.1 Independent Cascade Model

In the single-campaigner IC model, the campaign starts with an initial active (i.e., influenced) set of seed nodes, and then unfolds in discrete rounds. When some node u first becomes active at round t, it gets a single chance to activate each of its currently inactive out-neighbors v; it succeeds with probability $p_{u,v}$. If u succeeds, then v will become active at round $t+1$. Whether or not u succeeds at round t, it cannot make any further attempts in the subsequent rounds. If a node v has incoming edges from multiple newly activated nodes, their attempts are sequenced in an arbitrary order. Also, each node can be activated only once and it stays active until the end. The campaigning process runs until no more activations are possible.

Multi-Campaigner Independent Cascade Model. We shall now introduce the multi-campaigner Independent Cascade (MCIC) model [19], which models multiple campaigns that are being run simultaneously in a network. The MCIC model follows the same process as the IC model, except of two major differences. First, if some node u is activated with the campaign of C_i, it attempts to activate its out-neighbors v with the campaign of C_i. Second, an activated node v adopts one campaign uniform at random from all nodes u, which successfully activated node v in the last round. Each node can be activated only once and by only one

of the campaigns; also the node stays activated with that campaign until the end.

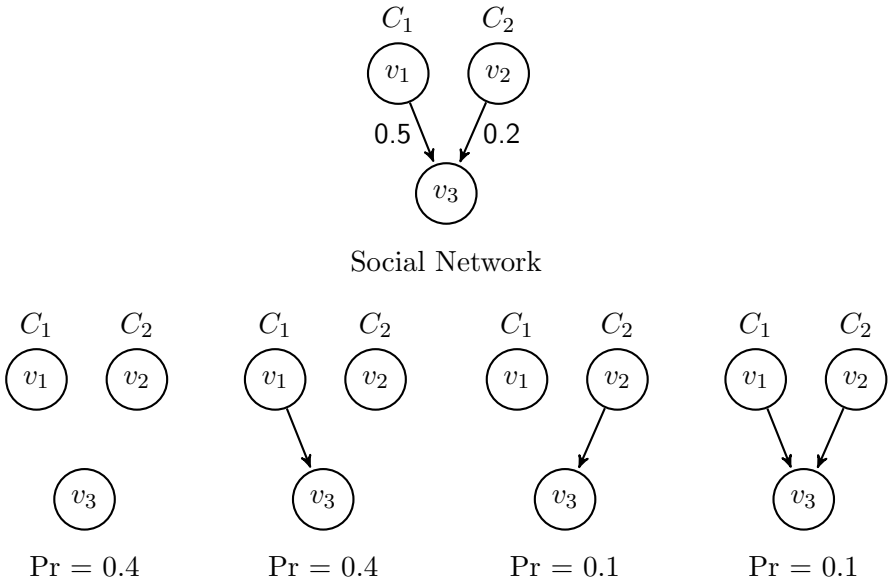


Figure 3.1: Example of the MCIC information diffusion model

Example 2 (MCIC Model) In Figure 3.1, we show a social network along with edge probabilities. We also assume that v_1 and v_2 are seed nodes for the campaigners C_1 and C_2, respectively, and we want to calculate the probability that node v_3 will be influenced by each of these campaigners using the MCIC information diffusion model. The computation is carried out as follows. We first identify all *possible worlds* corresponding to the uncertain social graph, where each possible world is a certain instance of the uncertain graph, and obtained by independent sampling of the edges. Each possible world is associated with a probability of existence. In our figure, v_3 is influenced by C_1 in the second possible world, by C_2 in the third possible world, and by either of C_1 and C_2 with equal probability in the fourth possible world. Therefore, the probability that v_3 is influenced by C_1 is $0.4 + 0.1 \times \frac{1}{2} = 0.45$, and that v_3 is influenced by C_2 with probability $0.1 + 0.1 \times \frac{1}{2} = 0.15$. It is important to note that the MCIC model assumes the following information cascading scenario: people adopt some product when they come in direct contact with their friends who very recently adopted that product. □

3.2.2 Linear Threshold Model

In the single-campaigner LT model, each node v has an activation threshold $\theta_v \leq 1$. In addition, there is a constraint that the sum of the probabilities of all incoming edges for every node must be at most 1. Kempe et al. [2] call the probabilities for the LT model weights instead.

Analogous to the IC model, the campaign starts with an initial active (i.e., influenced) set of seed nodes, and then unfolds in discrete rounds. If the sum of the probabilities of the incoming edges from all active nodes is greater or equal to the activation threshold of an inactive node, then the node gets activated in the next round. Each node can only be activated once and stays active until the end.

Multi-Campaigner Linear Threshold Model. The multi-campaigner LT model, also termed as the K-LT model in [3], follows two steps in each round. The first step decides whether a new node will get activated and it works in exactly the same way as the LT model (i.e., without distinguishing among multiple campaigns). However, in the second step, it decides which campaign each of those newly activated nodes will adopt. Let us consider all nodes u that were activated in the previous round and contributed to the activation of a node v in the current round. Then, v will adopt the same campaign as that of u with probability $\frac{p_{uv}}{\sum_u p_{uv}}$. With the K-LT model, each node can be activated only once and by only one of the campaigns; also the node stays activated with that campaign until the end.

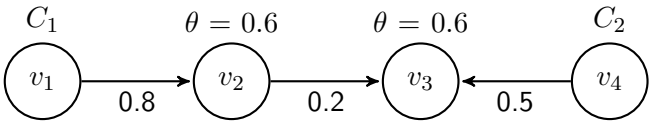


Figure 3.2: Example of the K-LT information diffusion model

Example 3 (K-LT Model) We demonstrate an example of the K-LT model in Figure 3.2. Assume that v_1 and v_4 are seed nodes for the campaigners C_1 and C_2, respectively. The K-LT model works as follows. At time step 1, v_2 becomes active with the campaign of C_1, since $p_{v_1,v_2} = 0.8 > \theta_{v_2} = 0.6$. However, v_3 remains inactive as $p_{v_4,v_3} = 0.5 < \theta_{v_3} = 0.6$. Subsequently, at time step 2, v_3 gets activated as the total incoming influence from its activated neighbors is: $p_{v_2,v_3} + p_{v_4,v_3} = 0.7$, which is higher than its activation threshold $\theta_{v_3} = 0.6$. Finally, v_3 selects the campaign of C_1 with probability 1, because v_2 is the only neighbor of v_3 which was activated in the last round, and v_2 was activated with the campaign of C_1. One may note that the K-LT model simulates the following scenario: a user adopts a technology only when more than a threshold number of its neighbors adopted a similar technology. However, once the user decides

to adopt the technology, she decides on the specific product only based on her neighbors who most recently adopted that technology. □

3.3 Hardness Results

In this section we proof that Problem 1 is **NP**-hard under both the MCIC and the K-LT information diffusion model. We consider the decision version of our problem, which is defined as follows. Given a social network $\mathcal{G} = (V, E, P)$, $m \geq 2$ client campaigners, the revenue matrix A, a budget k_i on the seed set size for each campaigner C_i, that is, $|S_i| = k_i$ for $1 \leq i \leq m$, and a positive integer R, can we find a seed set for each campaigner such that the expected revenue of the host is at least R?

Theorem 3.1 *Following the* MCIC *model of information diffusion, the decision version of Problem 1 is* **NP***-hard.*

Proof We shall prove the **NP**-hardness by performing a reduction from the **NP**-complete set-cover problem [22]. Let us consider an instance of the set-cover problem, defined by a collection of subsets $S = \{S_1, S_2, \ldots, S_r\}$ of a ground set $U = \{u_1, u_2, \ldots u_n\}$; we wish to know whether there exist k of the subsets whose union is equal to U. Usually, $k < r < n$. Now, we consider another identical instance of the previous set cover problem, given by a collection of subsets $S' = \{S'_1, S'_2, \ldots, S'_r\}$ of the ground set $U' = \{u'_1, u'_2, \ldots u'_n\}$. We construct our revenue maximization problem for $m = 2$ competing campaigners: C_1 and C_2, as follows. For each $u_i \in U$, $u'_i \in U'$, $S_i \in S$, and $S'_i \in S'$, we include a node in the network. The two nodes corresponding to element pairs u_i, u'_i are connected by a bi-directed edge of probability 1. If a subset S_i covers an element u_j, we add a directed edge of probability 1 from node S_i to node u_j in the network. Analogously, if some subset S'_i covers an element u'_j, we also add a directed edge of probability 1 from node S'_i to node u'_j in the network. The revenue matrix has the following form: $A_{1,u_i} = 1$ for all $u_i \in U$, $A_{2,u'_j} = 1$ for all $u'_j \in U'$, all other entries in the revenue matrix are 0. Finally, we also assume that there is a budget k on seed set size for each campaigner. In this setting, there is a solution to our revenue maximization problem with the host's expected revenue at least $2n$, if and only if there is a solution to the set cover problem. Hence, the theorem. ■

Theorem 3.2 *Following the* K-LT *model of information diffusion, the decision version of Problem 1 is* **NP***-hard.*

Proof We prove the **NP**-hardness by performing a reduction from the **NP**-complete set cover problem, defined by a collection of subsets $S = \{S_1, S_2, \ldots, S_r\}$

of a ground set $U = \{u_1, u_2, \dots u_n\}$; and we want to know whether there exist k of the subsets whose union is equal to U. Now, we construct our revenue maximization problem with the K-LT model and for $m = 2$ competing campaigners: C_1 and C_2, as follows. For each element $u_i \in U$, there is a node in the network with activation threshold $\frac{1}{r+1}$. For each subset $S_i \in S$, we add two nodes v_i and v_i' in the network, each having an activation threshold 1. Now, if a subset S_i covers an element u_j, we add a directed edge of probability $\frac{1}{2r}$ from node v_i to node u_j, and another directed edge of probability $\frac{1}{2r}$ from node v_i' to node u_j. Note that the sum of the probabilities of all incoming edges to any node u_j is at most 1. The revenue matrix A has the following form: $A_{1,u_i} = 1$ for all $u_i \in U$, $A_{2,u_i} = 1$ for all $u_i \in U$, and all other entries in A are 0. Finally, we also assume that there is a budget k on seed set size for each campaigner. In this setting, there is a solution to our revenue maximization problem with the host's expected revenue at least n, if and only if there is a solution to the set cover problem. Hence, the theorem. ∎

3.4 Monotonicity and Submodularity

The following two proofs by counterexample show that Problem 1 is neither monotonic nor submodular under both the MCIC model and the K-LT model. At the end of this section we show that Problem 1 adapted for one campaigner is submodular and monotonic, which is important for our baseline algorithms described in Section 3.5.1 and 3.6.1 and the first step of our solution for the K-LT model described in Section 3.6.2.

Theorem 3.3 *Problem 1 is not monotonic under both the* MCIC *and the* K-LT *model.* ◇

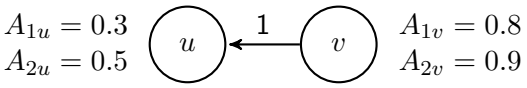


Figure 3.3: Counterexample of Monotonicity

Proof In Figure 3.3, we assume that node v is already assigned to the campaigner C_2 and that we still need to assign one seed node to the campaigner C_1. Under both the MCIC and the K-LT model node u is always activated by node v, if v is activated. Therefore under both models our revenue is $0.5 + 0.9 = 1.4$ with only node v assigned to the campaigner C_2. If we now assign node u to the seed set of the campaigner C_1 our revenue is reduced to $0.3 + 0.9 = 1.2$. If we assign the nodes in the other order, then our revenue is increased from 0.3 to $0.3 + 0.9 = 1.2$. Hence, the theorem. ∎

Theorem 3.4 *Problem 1 is not submodular under both the* MCIC *and the* K-LT *model. Formally, let* $F(S) = \underset{S_1,S_2,\ldots,S_m}{\arg\max} \sum_{i=1}^{m} \sum_{u \in V} [A_{iu} \cdot Pr(u, i, S)]$. *Then the following inequality does not always hold,*

$$F(S \cup \{v\}) - F(S) \geq F(S_1 \cup \{v\}) - F(S_1) \tag{3.2}$$

Here, $S_1 \supseteq S$ *and* $v \notin S_1$. ◇

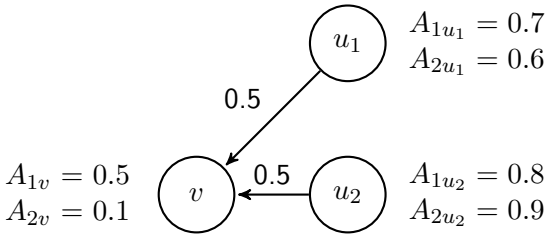


Figure 3.4: Counterexample of Submodularity

Proof In Figure 3.4, we assume that node u_1 is already assigned to the campaigner C_1 in S and that in S_1 we additionally have node u_2 assigned to the campaigner C_2. Under the MCIC model node v is always activated, if either node u_1 or u_2 is activated. Under the K-LT model we use an activation threshold of 0.4 for node v and edge probabilities, also called weights, of 0.5, therefore node v is always activated, if either node u_1 or u_2 is activated under both models. Now we assign v to the campaigner S_2 and check the submodularity inequality for S and S_1. So we get $F(S \cup \{v\}) - F(S) = (0.7 + 0.1) - (0.7 + 0.5) = -0.4$ and $F(S_1 \cup \{v\}) - F(S_1) = (0.7 + 0.9 + 0.1) - (0.7 + 0.9 + (0.5 * 0.5) + (0.1 * 0.5)) = -0.2$. Therefore the submodularity inequality does not hold and the theorem follows. ■

An adapted version of Problem 1 for one campaigner would actually be equal to Problem 2 with $m = 1$, which is monotonic and submodular as shown in Theorem 3.6. This allows our baseline algorithms described in Section 3.5.1 and 3.6.1 to apply iterative hill-climbing to achieve a performance guarantee of $1 - \frac{1}{e}$ for each individual campaigner, similar to [2].

3.5 Solution for the Independent Cascade Model

In this section, we first introduce a simple baseline solution to Problem 1 under the MCIC information diffusion model, which is a straightforward adaptation of the single-campaigner viral marketing techniques.

Although our revenue maximization problem is **NP**-hard over graphs, we illustrate next that the problem is solvable in polynomial time in a tree dataset. Therefore, we consider a two step heuristic approach for general directed graphs:

First, given a graph dataset, we extract the *most influential tree*, which is formally defined in Section 3.5.4. Intuitively, the most influential tree approximates a social network by preserving the *most influential path* between every pair of nodes as much as possible [23]. A path between a source and a destination node is called the most influential path if the probability of information diffusion along that path is maximal in comparison with all other paths between these two nodes. As empirically demonstrated in [23], the most influential paths often play a significant role in an information cascade in real-world social networks. Thus, we propose to approximate a social network by using its most influential tree.

Second, we design a polynomial-time exact algorithm to solve the revenue maximization problem over a tree dataset (Section 3.5.2). We describe our algorithm with a simpler binary tree in the following section, and later in Section 3.5.3, we show how to convert a tree to an equivalent binary tree suitable for our method.

3.5.1 Baseline Solution

This is a heuristic method with a straightforward adaptation of the single-campaigner viral marketing techniques. We process the campaigners in descending order of the aggregated money that each campaigner is willing to provide to the host for all the users in the network. We then identify the top-k seed nodes for each campaigner by an iterative hill-climbing algorithm such that the host's revenue is maximized by *considering one campaigner at a time*, similar to [2]. Nevertheless, in order to eliminate the information-cascading effect of already-selected seed nodes of previous campaigners, we delete these seed nodes from the graph before identifying the top-k seed nodes for the next campaigner.

3.5.2 Exact Solution for Directed Binary Trees

On a directed tree, a node v can only be activated by its closest ancestor u, including the node itself, such that u belongs to one of the seed sets S_1, S_2, ..., S_m. This is simply because u blocks the path from any farther ancestor u' to v, where u' is also a seed node. Therefore, $Pr(v, i, S)$, the probability that node v will be influenced by C_i's campaign, has the following expression in case of a directed tree.

$$Pr(v, i, S) = \begin{cases} 0 & \text{if } u \notin S_i; \\ \prod_{(u',v') \in Path(u,v)} p_{u'v'} & \text{otherwise.} \end{cases}$$

Here, u denotes the closest ancestor of v, such that u is a seed node. First, we compute for every node v in the tree dataset, the probability that v gets activated

by each of its ancestors u, which is simply $\prod_{(u',v')\in Path(u,v)} p_{u'v'}$. We store these activation probabilities in a table $\mathcal{B}$ of size $\mathcal{O}(nd)$, where n is the total number of nodes in the tree and d is the depth of the tree. Let $\mathcal{B}(u,v)$ denote the activation probability that node v is activated by its ancestor u. Clearly, $\mathcal{B}(v,v) = 1$. The computation of table $\mathcal{B}$ requires $\mathcal{O}(nd)$ time, if we use the activation probabilities from a parent node while computing the activation probabilities for its children nodes.

Next, we apply a dynamic-programming-based algorithm to find the optimal seed sets for all campaigners over a directed binary tree. For this purpose, we introduce another table OPT of the form $\mathsf{OPT}(v, u, j, [k'_1, k'_2, \ldots, k'_m]^T)$, where: **(a)** v is any node in the tree, **(b)** node u denotes the first ancestor of node v such that u is a seed node, **(c)** $j \in (1, m)$ denotes the campaigner C_j such that u is a seed node of the campaigner C_j, and **(d)** each $k'_i \leq k$ denotes the number of seed nodes already assigned to the campaigner C_i in the subtree rooted at v. An entry in the OPT table, e.g., $\mathsf{OPT}(v, u, j, [k'_1, k'_2, \ldots, k'_m]^T)$ represents the host's expected revenue for the optimal assignment of all seed sets S_i, $i \in (1, m)$, $|S_i| = k'_i$ over the subtree rooted at node v, given u, which is an ancestor of v, is assigned as a seed node to the campaigner C_j. It can be noted that the size of OPT table is $\Theta(ndmk^m)$.

The entries in OPT are computed by performing a post-order traversal over the tree dataset as given below.

$$\mathsf{OPT}\left(v, u, j, \begin{bmatrix} k'_1 \\ \vdots \\ k'_m \end{bmatrix}\right) = \max\{F_1, F_2\} \tag{3.3}$$

$$F_1 = \max_{k''_1=0}^{k'_1} \max_{\cdots}^{\cdots} \max_{k''_m=0}^{k'_m} \left\{ \mathsf{OPT}\left(l(v), u, j, \begin{bmatrix} k''_1 \\ \vdots \\ k''_m \end{bmatrix}\right) + \mathsf{OPT}\left(r(v), u, j, \begin{bmatrix} k'_1 - k''_1 \\ \vdots \\ k'_m - k''_m \end{bmatrix}\right) + A_{j,v} \times \mathcal{B}(u,v) \right\} \tag{3.4}$$

$$F_2 = \max_{i=1}^{m} \left\{ \max_{k_1''=0}^{k_1'} .. \max_{k_i''=0}^{k_i'-1} .. \max_{k_m''=0}^{k_m'} \left\{ \mathsf{OPT} \left(l(v), v, i, \begin{bmatrix} k_1'' \\ \vdots k_i'' \\ \vdots \\ k_m'' \end{bmatrix} \right) \right. \right.$$
$$\left. \left. + \mathsf{OPT} \left(r(v), v, i, \begin{bmatrix} k_1' - k_1'' \\ \vdots \\ k_1' - k_i'' - 1 \\ \vdots \\ k_m' - k_m'' \end{bmatrix} \right) + A_{i,v} \right\} \tag{3.5}$$

In the aforementioned dynamic programming formulation, the value of function $\mathsf{OPT}(v, u, j, [k_1', k_2', \ldots, k_m']^T)$ is computed as the maximum over two cases. **(a)** F_1: the first case considers the scenario when v is not selected as a seed node, and **(b)** F_2: the second case considers the situation when v is assigned to some campaigner as a seed node. Here, $l(v)$ and $r(v)$ denote the left and right subtrees of v, respectively, as illustrated in Figure 3.5. For simplicity of description, let us assume that the budget of the seed set size for each of the m campaigners is k. Then, to fill one entry in the OPT table, we need $\mathcal{O}(mk^m)$ time. Therefore, the time complexity of our dynamic programming is $\mathcal{O}(ndm^2k^{2m})$.

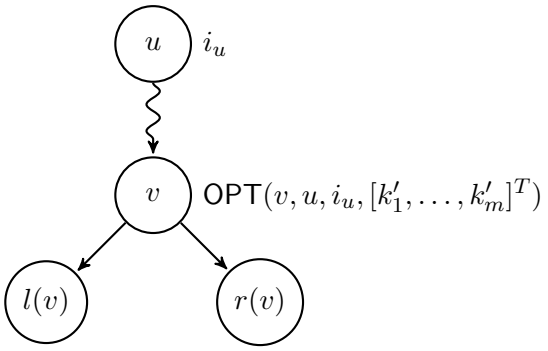


Figure 3.5: Exact Solution over Binary Tree: MCIC model

It is important to note that the dynamic programming terminates by computing the OPT entries for the root node v_r, that is, $\mathsf{OPT}(v_r, -1, i, [k, \ldots, k]^T)$ and $\mathsf{OPT}(v_r, v_r, i, [k, \ldots, k]^T)$ for all $i \in (1, m)$. The value -1 at the second index indicates that the root node v_r does not have an ancestor. Therefore, $\mathsf{OPT}(v_r, -1, i, [k, \ldots, k]^T)$ is invariant of i. Once we terminate our dynamic programming, we determine whether a node will be a seed node (and if so, to which campaigner that node will be assigned) by *backtracking* using the OPT entries of its child nodes. The backtracking process requires another $\mathcal{O}(mnk^m)$ time.

Therefore, the overall time complexity of finding the optimal seed nodes over a directed binary tree is $\mathcal{O}(ndm^2k^{2m})$. We note that our dynamic-programming-based exact solution over directed binary trees has a polynomial-time complexity in the number of tree nodes; however, it has an exponential time-complexity in the number of client campaigners.

Space and Time Complexity. In this section, we summarize the space and time complexity of our dynamic-programming-based solution. The space complexity is $\mathcal{O}(ndmk^m)$: table $\mathcal{B}$ has size $\mathcal{O}(nd)$ and table OPT has size $\Theta(ndmk^m)$. The time complexity of our dynamic programming is $\mathcal{O}(ndm^2k^{2m})$.

3.5.3 Conversion from Directed Trees to Binary Directed Trees

Our dynamic-programming-based exact solution for the revenue maximization problem (Problem 1) is also applicable over non-binary trees. In fact, given a directed tree, we first convert it to an equivalent directed binary tree. We use the conversion technique as proposed in [7]. For each non-leaf node v with children $v_1, v_2, \ldots, v_\Delta$, where $\Delta > 2$ in the original tree, we replace v with a binary tree of depth at most $\log \Delta$ and leaves $v_1, v_2, \ldots, v_\Delta$. For each newly introduced node u, we assign the revenue $A_{i,u} = 0$, for all campaigners $i \in (1, m)$. While applying our dynamic programming as discussed in Section 3.5.2, we also incur an additional constraint that no such newly introduced node can be selected as a seed node. Finally, for each newly introduced edge, the direction is always from the root towards the leaves, and each of them has probability 1. The incoming edges to the leaves $v_1, v_2, \ldots, v_\Delta$ have the same probability as that of the previous incoming edges to nodes $v_1, v_2, \ldots, v_\Delta$, respectively. This conversion process ensures that the newly introduced edges and nodes will not affect the MCIC propagation model as in the original directed tree.

As shown in [7], for the aforementioned *tree to binary tree* conversion method, the number of nodes in the equivalent binary tree is at most twice the number of nodes in the directed input tree, as well as the depth of the binary tree is at most a factor of $\log \Delta^*$ larger than the depth of the original tree, where Δ^* is the maximum out-degree of any node in the input tree.

3.5.4 Extraction of the Most Influential Directed Tree from a general Graph

The revenue maximization problem is **NP**-hard in directed graphs as shown in Theorem 3.2. Therefore, given a directed and connected graph G, we first extract the *most influential* tree T^* similar to [7], which is a directed spanning tree of G, and formally defined below.

Definition 3.5 (Most Influential Tree) *Given a general connected directed graph $G = (V, E, P)$ with a root node v_r, the most influential tree $T^* = (V, E_{T^*}, P)$ with $E_{T^*} \subseteq E$ is a directed spanning tree of G, with the same root node v_r, such that the product of the edge probabilities in T^* is maximized. Formally,*

$$T^* = \underset{T \in SpanningTree(G)}{\arg\max} \prod_{(u,v) \in E_T} p_{u,v} \tag{3.6}$$

◇

Intuitively, the most influential tree aims at preserving the most influential path between every pair of nodes as much as possible. These most influential paths play an important role in the information cascade over real-world social networks [23].

We note that the problem of finding the most influential tree can be converted to the problem of finding the minimum-cost *directed* spanning tree by minimizing the summation of negative logarithms of the edge probabilities in T^* as given in Equation 3.7 instead of using Equation 3.6.

$$T^* = \underset{T \in \text{SpanningTree}(G)}{\arg\min} \sum_{(u,v) \in E_T} -\log(p_{u,v}) \tag{3.7}$$

Thus, one can find the most influential directed tree in time $\mathcal{O}(e + n \log n)$ due to Gabow et al. [24]. It is important to note that [24] requires some root node v_r to be present in the input graph G such that all other nodes in G are reachable from v_r. If there is no such root node present in G, we add a dummy root node v_r as follows [7]: find the set $V_r \subseteq V$ of nodes in G which do not have any incoming edges. We then connect these nodes to a dummy root node v_r, with edges directed towards the nodes in V_r and each of these newly-introduced edges is assigned a very low edge-probability. Now, all nodes which are not reachable from v_r are part of a ring of nodes. From each such ring of nodes we add an arbitrary node to V_r and connect v_r to those nodes with an edge as above. Finally, all nodes are reachable form v_r.

Because v_r is the dummy root node, we also assign the revenue $A_{i,v_r} = 0$ for all campaigners $i \in (1, m)$; and then, we further incur an additional constraint that the dummy root node v_r cannot be selected as a seed node during our dynamic-programming-based exact solution over the most influential tree T^*.

3.6 Solution for the Linear Threshold Model

In this section, we consider the revenue maximization problem (Problem 1) under the K-LT information diffusion model. We first introduce a simple baseline solution and then present our solutions to the problem.

We recall that our problem is **NP**-hard under the K-LT model (Theorem 3.2). However, we shall later illustrate that given an *already-selected* set of seed nodes,

one can optimally partition these seed nodes among m campaigners in polynomial time such that the host's expected revenue is maximized. Therefore, we design a two-step heuristic technique to solve our original revenue maximization problem with a *theoretical performance guarantee* of $\frac{1}{m}(1-\frac{1}{e})$, where m is the number of campaigners.

In the first phase (Section 3.6.2), the host *optimistically* assumes that it is possible to influence each user by a campaign such that the corresponding campaigner spends the maximum amount of money for that particular user. This is equivalent to assigning, for each user u, a revenue A_u which is the maximum of $A_{i,u}$ values over all campaigners i. Thus, the host identifies km seed nodes assuming there is only *one campaigner* and with the objective that her expected revenue is maximized under this optimistic assumption. However, we recall that, in reality, there are m campaigners, each with a seed set of size k.

Therefore, in the second step (Section 3.6.3), the host partitions these *already-selected* km seeds among m campaigners with the objective that her expected revenue is maximized in the actual *multi-campaigner* setting and considering the original revenue matrix. Below, we describe both these steps in detail. As this second step is still exponential in the number of campaigners, we also design a greedy version for Problem 3, which achieves a *theoretical performance guarantee* of $\frac{1}{2}$ and is described in Section 3.6.4.

For simplicity, we assume that the budget of the seed set size for each of the m campaigners is k.

3.6.1 Baseline Solution

This is the same heuristic method with a straightforward adaptation of the single-campaigner viral marketing techniques, as the one for the IC information diffusion model described in Section 3.5.1. The only difference is that we use the LT information diffusion model.

3.6.2 Optimistic Seed Set Selection

In the first phase, the host optimistically assumes that each user in the network can be influenced by a campaign such that the corresponding campaigner spends the maximum amount of money for that particular user. In other words, for each user u in the network, the host optimistically assigns a revenue A_u which is the maximum of $A_{i,u}$ values over all campaigners i. Formally, $A_u = \max_{i \in (1,m)} \{A_{i,u}\}$. Therefore, the host solves the following problem in the first step.

Problem 2 (Optimistic Seed Set Selection) Assuming that there is only one campaigner and given a revenue A_u for each user u in the network, find the seed set of size km such that the expected revenue of the host is maximized. Formally,

$$\begin{aligned} &\arg\max_{S} \sum_{u \in V} [A_u \cdot Pr_{\mathsf{LT}}(u, S)] \\ &\text{such that} \quad |S| = km \end{aligned} \tag{3.8}$$

Here, $Pr_{\mathsf{LT}}(u, S)$ denotes the expected spread of an information from the seed set S to node u following the classic Linear threshold (LT) model with one campaigner. Unfortunately, Problem 2 is also **NP**-hard following [2]; nevertheless, the objective function is monotonic and submodular as shown in Theorem 3.6.

Theorem 3.6 *The objective function of Problem 2 is submodular. Formally, let* $F(S) = \sum_{u \in V} [A_u \cdot Pr_{\mathsf{LT}}(u, S)]$. *Then,*

$$F(S \cup \{v\}) - F(S) \geq F(S_1 \cup \{v\}) - F(S_1) \tag{3.9}$$

Here, $S_1 \supseteq S$ *and* $v \notin S_1$. ◇

Proof The proof follows by considering the *live-edge* model, which is shown to be equivalent to the LT model in [2]. In the live-edge model, each node v picks at most one of its incoming edges at random, that is, it selects the incoming edge from u with probability $p_{u,v}$, and it does not select any incoming edge with probability $1 - \sum_{u \in in(v)} p_{u,v}$. Let X be one possible world with probability $Prob(X)$ under the live-edge model, and $R_X(S)$ be the host's revenue due to nodes that are reachable from the seed set S in that possible world X. One may verify that $R_X(S)$ is submodular with respect to S. Now, our objective function $F(S)$ is given by:

$$F(S) = \sum_{\text{all possible world } X} [Prob(X) \cdot R_X(S)] \tag{3.10}$$

As the non-negative linear combination of submodular functions is also submodular, $F(S)$ is submodular. ■

Thus, we apply an iterative hill-climbing algorithm (Algorithm 1) that finds the seed set with approximation guarantee $(1 - \frac{1}{e})$ of the optimal solution [25]. The hill-climbing algorithm works in km iterative steps. At each iteration, the algorithm selects a non-seed node u as a seed node, such that the expected revenue due to u and the previously selected seed nodes is maximized. Our hill-climbing-based iterative solution for the optimistic seed selection problem

Algorithm 1 Hill-Climbing Algorithm for the Optimistic Seed Set Selection

Require: Graph $G = (V, E, P)$, $A_u = \max_i\{A_{i,u}\}\ \forall u \in V$
Ensure: seed set S of size km
1: $S = \phi$
2: **for** $i = 1$ **to** km **do**
3: $\quad v = \arg\max_{v \in V} F(S \bigcup \{v\})$ $\quad$ / / $F()$ is defined in Theorem 3.6
4: $\quad S = S \bigcup \{v\}$
5: **end for**
6: output S

(Problem 2) is similar to state-of-the-art viral marketing techniques that identify the top-k seed nodes for a single campaigner such that its expected information spread in the network is maximized. Although we optimize the host's expected revenue instead of the expected information spread, due to the single-campaigner and submodular nature of Problem 2, one can easily apply an existing viral marketing algorithm (with some modification in the objective function) as the underlying technique to solve Problem 2.

We shall later show in Theorem 3.8 that the iterative hill-climbing algorithm for the optimistic seed selection, coupled with an optimal partition of that seed set, generates a solution to the original revenue maximization problem with the approximation guarantee $\frac{1}{m}(1 - \frac{1}{e})$, where m is the number of the campaigners.

Time Complexity. The time complexity of our iterative hill climbing algorithm is $\mathcal{O}(kmn(n+e)t)$, where km is the total number of seed nodes identified, and t is the number of Monte-Carlo samples performed over the entire graph in order to find one seed node. As one can apply state-of-the-art viral marketing techniques to solve Problem 2, we use the CELF++ algorithm [9], which further improves the efficiency based on smart pruning techniques.

3.6.3 Optimal Partition of the Seed Set

In the second phase, the host optimally partitions the previously selected km seed nodes among m campaigners, such that her expected revenue is maximized under the actual *multi-campaigner* K-LT model and considering the original revenue matrix. We formally define our problem statement for the second step as follows.

Problem 3 (Optimal Seed-Set-Partition) Given the already selected seed set S of size km and the revenue matrix $(A_{iu})_{m \times n}$, partition the seed set S into m subsets $S_1, S_2, \ldots, S_m$, such that each S_i has size k, and the expected revenue of the host is maximized following the multi-campaigner K-LT model.

We show that Problem 3 can be solved optimally in polynomial time using a dynamic-programming-based approach. For this purpose, we introduce the notion of *individual revenue* of the host from individual seed nodes.

Definition 3.7 (Individual Revenue) *The* individual revenue $\mathcal{R}_i(u)$ *represents the expected revenue of the host from a seed node $u \in S$ when u is assigned to the i-th campaigner C_i.* ◇

Individual Revenue Computation. We now describe our method to compute individual revenues. We start by randomly assigning a distinct number from 1 to km to every seed node in S. Let us denote by $I(u)$ the number assigned to seed node u. We also associate a list $\mathcal{L}$ of size km with each node v in the network. The j-th entry of list $\mathcal{L}(v)$, denoted as $\mathcal{L}_j(v)$, represents the spread that some seed node $u \in S$ can achieve at node v following the K-LT model, where $I(u) = j$. For a seed node $u \in S$, we initialize:

$$\mathcal{L}_j(u) = \begin{cases} 1, & \text{if } I(u) = j; \\ 0, & \text{otherwise.} \end{cases}$$

Next, we simulate the K-LT model starting from seed nodes in S. At any discrete step of the K-LT model, if some node v becomes active, we consider all its in-neighbors $v' \in in(v)$ that were activated in the previous step. We compute $\mathcal{L}_j(v)$ as follows:

$$\mathcal{L}_j(v) = \frac{\sum\limits_{\substack{v' \in in(v) \\ v' \text{activated in prev. step}}} [p_{v'v} \times \mathcal{L}_j(v')]}{\sum\limits_{\substack{v' \in in(v) \\ v' \text{activated in prev. step}}} p_{v'v}} \tag{3.11}$$

Finally, we compute the individual revenue $\mathcal{R}_i(u)$ for every seed node $u \in S$ and for every campaigner C_i as given in Equation 3.12.

$$\mathcal{R}_i(u) = \sum_{v \in V} [A_{iv} \times \mathcal{L}_{I(u)}(v)] \tag{3.12}$$

We refer to $\mathcal{R}_i(u)$ as the individual revenue of the host due to seed node u, when u is assigned to the campaigner C_i. We demonstrate the computation of individual revenues with an example below.

Example 4 In Figure 3.6, we assume there are three seed nodes: u_1, u_2, and u_3, and also two campaigners: C_1 and C_2. The seed nodes are not assigned to any specific campaigners yet. In the beginning, all seed nodes are activated, and in the next round, v gets activated, since its activation threshold $\theta_v = 0.6 < p_{u_1,v} + p_{u_2,v} + p_{u_3,v} = 0.9$. Following Equation 3.11, we get: $\mathcal{L}_{u_1}(v) = \frac{0.4}{0.4+0.3+0.2} = \frac{4}{9}$. Similarly, $\mathcal{L}_{u_2}(v) = \frac{3}{9}$, and $\mathcal{L}_{u_3}(v) = \frac{2}{9}$. Finally, we compute the individual revenues by following Equation 3.12. For example, $\mathcal{R}_1(u_1) = \sum_{v=u_1,u_2,u_3,v} A_{1,v} \times \mathcal{L}_{u_1}(v) = 1 \times 1 + 0 + 0 + 0.5 \times \frac{4}{9} = 1.22$. Similarly, we have: $\mathcal{R}_1(u_2) = 0 + 0 + 0 + 0.5 \times \frac{3}{9} = 0.17$. □

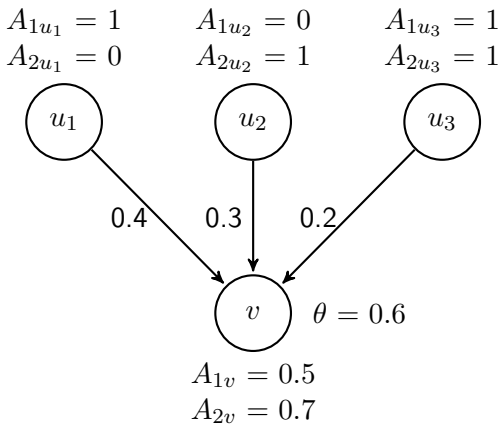


Figure 3.6: Example of Individual Revenue Computation

The following equations show the connection between Problem 1 and the individual revenue:

$$\begin{aligned}
&\arg\max_{S_1,S_2,\ldots,S_m} \sum_{i=1}^{m} \sum_{u\in V} [A_{iu} \times Pr(u,i,S)] \\
=\ &\arg\max_{S_1,S_2,\ldots,S_m} \sum_{i=1}^{m} \sum_{u\in V} [A_{iu} \times \sum_{v\in S_i} \mathcal{L}_{I(v)}(u)] \\
=\ &\arg\max_{S_1,S_2,\ldots,S_m} \sum_{i=1}^{m} \sum_{v\in S_i} \sum_{u\in V} [A_{iu} \times \mathcal{L}_{I(v)}(u)] \\
=\ &\arg\max_{S_1,S_2,\ldots,S_m} \sum_{i=1}^{m} \sum_{v\in S_i} \mathcal{R}_i(v)
\end{aligned} \tag{3.13}$$

The first equality holds, because only the seed nodes in S_i can influence node u to adapt the campaign of C_i. Therefore the summation of $\mathcal{L}_{I(v)}(u)$ over all seed nodes v of the campaigner C_i equals the probability, that node u will adapt the campaign of C_i under the seed set S. The last summation could be used as part of an alternative definition of Problem 1, which is based on the individual revenue instead of the revenue matrix A.

Properties of Individual Revenue. The individual revenue $\mathcal{R}_i(u)$ satisfies several interesting properties which are critical for our dynamic-programming-based exact solution.

Proposition 1 *For a given seed node u and a given campaigner C_i, $1 \leq i \leq m$, $\mathcal{R}_i(u)$ is invariant to how other seed nodes are assigned to various other campaigners.* □

Proposition 2 *If a set of seed nodes $u_1, u_2, \ldots, u_j \in S$ are all assigned to a given campaigner C_i, $1 \leq i \leq m$, then the expected revenue of the host due to $u_1, u_2, \ldots, u_j$ is simply the aggregation of $\mathcal{R}_i(u_1), \mathcal{R}_i(u_2), \ldots, \mathcal{R}_i(u_j)$; and this is invariant to how the remaining seed nodes are assigned to other campaigners.* □

The first proposition directly follows from the definition of the K-LT model, given a pre-defined seed set S, the activation of other nodes in the network is determined by the first phase of the K-LT model, that is, the classical LT model assuming all the campaigners are cascading the same information. More specifically, if a node in the network will be activated or not is independent of how S is partitioned among multiple campaigners [3]. The partition of S only influences the following: what campaign each active node in the network will adapt and with how much probability. Equation 3.13 also shows the correctness of both propositions, because if some seed nodes change their seed set, the sum of the individual revenues of the other seed nodes remains the same. The second proposition follows from the linearity property of $\mathcal{L}_j(v)$ in Equation 3.11, that is, $\mathcal{L}_{\{u_1,u_2\}}(v) = \mathcal{L}_{u_1}(v) + \mathcal{L}_{u_2}(v)$, for any two distinct seed nodes $u_1, u_2 \in S$ and for any node v in the graph.

Example 5 In Figure 3.6, $\mathcal{R}_1(u_1) = 1.22$ and $\mathcal{R}_1(u_2) = 0.17$. We note that, $\mathcal{R}_1(u_1)+\mathcal{R}_1(u_2) = 1.39$, and this is exactly the same as $\mathcal{R}_1(\{u_1, u_2\})$, which is the expected revenue of the host, when both u_1 and u_2 are assigned to campaigner 1. □

Dynamic Programming Based Exact Solution. We are now ready to describe our dynamic-programming-based exact algorithm to solve the optimal seed-set-partitioning problem (Problem 3). Our algorithm processes the seed nodes $u \in S$ in the ascending order of their assigned $I(u)$ numbers. We recall that $I(u) \in (1, km)$. The dynamic programming algorithm maintains a table $\mathsf{EXACT}(j, [k'_1, k'_2, \ldots, k'_m]^T)$, that stores the optimal expected revenue of the host when we have already partitioned the seed nodes with numbers from 1 to j into m subsets $\{S_1, S_2, \ldots, S_m\}$, such that $|S_i| = k'_i \leq k$, and we have also assigned them to the respective campaigners. Clearly, $1 \leq j \leq km$ and $j = \sum_{i=1}^{m} k'_i$. The dynamic programming proceeds as given below.

$$\mathsf{EXACT}\left(j, \begin{bmatrix} k'_1 \\ k'_2 \\ \vdots \\ k'_m \end{bmatrix}\right) = \max_{i \in (1,m)} \left\{ \mathsf{EXACT}\left(j-1, \begin{bmatrix} k'_1 \\ \vdots \\ k'_i - 1 \\ \vdots \\ k'_m \end{bmatrix}\right) + \mathcal{R}_i(u) \right\} \tag{3.14}$$

We note that in the recursive equation above, u denotes the node with $I(u) = j$. The optimal assignment of the last seed node is determined by

$\mathsf{EXACT}(km, [k, k, \ldots, k]^T)$. The optimal assignment of the previous seed nodes is determined by *backtracking* with the usage of the EXACT values. The correctness of our dynamic-programming-based solution follows from Propositions 1 and 2.

Space and Time Complexity. Our algorithm has space complexity $\mathcal{O}(k^m)$ due to the EXACT table. The time complexity of our dynamic programming is $\mathcal{O}(mk^m)$. This is because we need to fill the table of size k^m; and in order to fill each entry in the table, we compute the maximum of m values. The backtracking requires another $\mathcal{O}(m^2k)$ time, since there are km seed nodes that we need to assign to different campaigners, and for each seed node, we require to compare m values to find the best assignment. In addition, one needs to compute the expected revenue vectors for all seed nodes by running a breadth-first-search from each of these km seed nodes. Hence, the time required to compute the individual revenue vectors for all seed nodes is $\mathcal{O}(km(n+e))$, where n and e are the number of nodes and edges in the graph, respectively. Therefore, the time complexity of our exact solution is $\mathcal{O}(kmn + ekm + m^2k + mk^m)$. We note that our optimal seed-set-partitioning solution requires polynomial time in the size of the graph.

Performance Guarantee. Theorem 3.8 provides the overall *approximation guarantee* of our method for the host's revenue maximization problem under the $\mathsf{K\text{-}LT}$ model.

Theorem 3.8 *The iterative hill-climbing solution of the optimistic seed selection problem (Problem 2), coupled with the optimal partition of that seed set (Problem 3), guarantees* $\frac{1}{m}(1 - \frac{1}{e})$ *approximation to the original revenue maximization problem (Problem 1) under the* $\mathsf{K\text{-}LT}$ *model, with the assumption that each campaigner has the same number of seed nodes. Here,* m *is the number of campaigners.*

Proof Let us denote by S_{opt} the optimal seed sets for the revenue maximization problem (Problem 1) under the $\mathsf{K\text{-}LT}$ model. Then, the host's maximum revenue can be written as: $\sum_{u \in V} \sum_{i=1}^{m} [A_{i,u} \times Pr_{\mathsf{K-LT}}(u, i, S_{opt})]$. Here, $Pr_{\mathsf{K-LT}}(u, i, S_{opt})$ denotes the expected spread of the campaign C_i from the seed sets S_{opt} to a node u following the $\mathsf{K\text{-}LT}$ model.

Next, let us denote by S^* the best seed sets for the optimistic seed selection problem (Problem 2). It is easy to verify that

$$\sum_{u \in V} \sum_{i=1}^{m} [A_{i,u} \times Pr_{\mathsf{K-LT}}(u, i, S_{opt})] \leq \sum_{u \in V} [A_u \times Pr_{\mathsf{LT}}(u, S^*)] \tag{3.15}$$

This is because $A_u = \max_{i \in (1,m)} A_{i,u}$. Hence, the left hand side of the inequality must be at most $\sum_{u \in V} [A_u \times \sum_{i=1}^{m} Pr_{\mathsf{K-LT}}(u, i, S_{opt})]$, which is equal to

$\sum_{u \in V}[A_u \times Pr_{\mathsf{LT}}(u, S_{opt})]$. The equality in the last step follows from the definition of the K-LT model, that is, the activation probability of a node under the K-LT model is independent of how the seed set S_{opt} is partitioned among multiple campaigners. Hence, if Inequality 3.15 were not true, then S^* is not the optimal solution to Problem 2, as clearly a better solution S_{opt} for Problem 2 exists, which is a contradiction.

Let us define $V_i \subseteq V$ such that for each node $u \in V_i : A_u = A_{i,u}$ and $\forall i \neq j : V_i \cap V_j = \emptyset$ and $\bigcup_{i=1}^{m} V_i = V$. The following holds, because the summation still goes over all nodes in V.

$$\begin{aligned}
&\sum_{u \in V}[A_u \times Pr_{\mathsf{LT}}(u, S^*)] \\
=&\sum_{i=1}^{m}\sum_{u \in V_i}[A_u \times Pr_{\mathsf{LT}}(u, S^*)] \\
=&\sum_{i=1}^{m}\sum_{u \in V_i}[A_{i,u} \times Pr_{\mathsf{LT}}(u, S^*)] \\
\leq&\sum_{i=1}^{m}\left(\sum_{u \in V}[A_{i,u} \times Pr_{\mathsf{LT}}(u, S^*)]\right) \\
=&\sum_{i=1}^{m}\left(\sum_{w \in S^*}\mathcal{R}_i(w)\right)
\end{aligned} \tag{3.16}$$

The equality in the last line follows from the definition of $R_i(w)$, as the right side of the equality, i.e., $\sum_{w \in S^*}\mathcal{R}_i(w)$ means that we assign the whole seed set S^* to the campaigner C_i.

Next, let us denote by $\mathcal{R}_{\mathsf{K-LT}}$ the host's expected revenue corresponding to the optimal partition of S^*. Since we assume that each campaigner has the same number of seed nodes, one may verify that $\mathcal{R}_{\mathsf{K-LT}} \geq \frac{1}{m}\sum_{i=1}^{m}\sum_{w \in S^*}\mathcal{R}_i(w)$ always holds. The left hand side and the right hand side of the inequality are equal if and only if the individual revenue vectors are equal, that is, $\forall w \in S^*$ and $\forall i, j \in (1, m)$, it holds that $R_i(w) = R_j(w)$. Here, m is the number of campaigners.

By combining Inequalities 3.15 and 3.16, we get:

$$\mathcal{R}_{\mathsf{K-LT}} \geq \frac{1}{m}\sum_{u \in V}\sum_{i=1}^{m}[A_{i,u} \times Pr_{\mathsf{K-LT}}(u, i, S_{opt})] \tag{3.17}$$

Finally, considering the fact that the optimal selection problem (Problem 2) is **NP**-hard and our iterative hill-climbing method produces a solution which is

at least $(1 - \frac{1}{e})$ of the solution corresponding to the optimal seed set S^*, the overall approximation ratio of our method is given by $\frac{1}{m}(1 - \frac{1}{e})$. Hence, the theorem. ∎

3.6.4 Greedy Partition of the Seed Set

Our dynamic-programming-based exact solution for the seed-set-partitioning problem (Problem 3) is still exponential in the number of campaigners. Therefore, we also design an efficient greedy algorithm to solve Problem 3 with a theoretical performance guarantee of $\frac{1}{2}$ to the optimal partition.

Greedy Partitioning Algorithm. Our greedy partitioning technique is given in Algorithm 2. For each seed node $u \in S$, we define the *individual revenue vector* $\mathcal{R}(u)$ as the collection of individual revenues $\mathcal{R}_i(u)$, for all campaigners C_i. Formally, $\mathcal{R}(u) = \{\mathcal{R}_i(u) : i \in (1, m)\}$. Next, for each seed node u, we find the dimension i_1 of its individual revenue vector $\mathcal{R}(u)$ that contains the largest value, among all other values present in other dimensions of $\mathcal{R}(u)$. We also use a max-heap $\mathbb{L}$ to store the tuples of the form $\langle u, i_1 \rangle$ in descending order of $\mathcal{R}_{i_1}(u)$ values. Next, we keep on retrieving and deleting the top elements from $\mathbb{L}$ in order until $\mathbb{L}$ is empty. At any point, if $\langle u, i_1 \rangle$ is the current tuple retrieved from the top of $\mathbb{L}$, and the size of S_{i_1} is not yet k, then we greedily assign seed node u to S_{i_1}. Otherwise, if S_{i_1} is full, we find the dimension i_2 of $\mathcal{R}(u)$ that contains the largest value such that the size of S_{i_2} is still less than k. We then insert $\langle u, i_2 \rangle$ into $\mathbb{L}$ based on the value in $\mathcal{R}_{i_2}(u)$. We terminate our algorithm when $\mathbb{L}$ is empty, and then each S_{i_1} is assigned as the seed set of the campaigner C_{i_1}.

Example 6 We illustrate our greedy partitioning method (Algorithm 2) with an example. Assume there are 3 campaigners and total 6 seed nodes with individual revenue vectors as follows: $\mathcal{R}(u_1) = [20\ 15\ 3]$, $\mathcal{R}(u_2) = [19\ 17\ 18]$, $\mathcal{R}(u_3) = [18\ 9\ 10]$, $\mathcal{R}(u_4) = [17\ 12\ 30]$, $\mathcal{R}(u_5) = [16\ 1\ 9]$, and $\mathcal{R}(u_6) = [15\ 2\ 7]$. We also assume that each campaigner will be assigned 2 seed nodes. Our greedy algorithm initially stores the seed notes in $\mathbb{L}$ in the following order: $\langle u_4, 3 \rangle$, $\langle u_1, 1 \rangle$, $\langle u_2, 1 \rangle$, $\langle u_3, 1 \rangle$, $\langle u_5, 1 \rangle$, and $\langle u_6, 1 \rangle$. As we access and delete the top elements from $\mathbb{L}$, the first three nodes will be assigned to the best possible campaigner for each of them, that is, u_4 is assigned to the campaigner C_3, while u_1 and u_2 are both assigned to the campaigner C_1. At this point, the top element from $\mathbb{L}$ is $\langle u_3, 1 \rangle$; however, we cannot assign u_3 to the campaigner C_1, as C_1 already has 2 seed nodes. Therefore, we shall re-insert $\langle u_3, 3 \rangle$ with value 10 in the max-heap $\mathbb{L}$. Similarly, nodes u_5 and u_6 get re-inserted as $\langle u_5, 3 \rangle$ and $\langle u_6, 3 \rangle$ with values 9 and 7, respectively, into $\mathbb{L}$. By following the steps of the greedy algorithm, we shall eventually assign u_3 to the campaigner C_3, while both b_5 and b_6 get assigned to the campaigner C_2.

The expected revenue of the host by following our greedy algorithm is 82, while the best possible expected revenue from these seed nodes is 104, which

Algorithm 2 Greedy Partition of the Seed Sets

Require: seed set S of size km, individual revenue vector $\mathcal{R}(u)$ for each seed node $u \in S$
Ensure: partition S into m equal-sized subsets: $S_1, S_2, \ldots, S_m$
1: $S_i = \phi$, for all $i \in (1, m)$
2: sorted $\mathbb{L} = \phi$ [max-heap]
3: **for all** $u \in S$ **do**
4: $i_1 = \arg\max_{i \in (1,m)} \{\mathcal{R}_i(u)\}$
5: insert $\langle u, i_1 \rangle$ into $\mathbb{L}$ in descending order of $\mathcal{R}_{i_1}(u)$ values
6: **end for**
7: **while** $\mathbb{L}$ not empty **do**
8: $\langle u, i_1 \rangle = Top(\mathbb{L})$ [get and delete the top element of $\mathbb{L}$]
9: **if** S_{i_1} not full **then**
10: insert u into S_{i_1}
11: **else**
12: $i_2 = \arg\max_{i \in (1,m)} \{\mathcal{R}_i(u) : S_{i_2} \text{ not full}\}$
13: insert $\langle u, i_2 \rangle$ into $\mathbb{L}$ based on $\mathcal{R}_{i_2}(u)$ value
14: **end if**
15: **end while**

corresponds to the following optimal assignment: u_3, u_6 are assigned to C_1, u_1, u_2 to C_2, and u_4, u_5 to C_3. □

Performance Guarantee. Next, we shall prove that our greedy algorithm ensures a theoretical performance guarantee of $\frac{1}{2}$ to the optimal solution.

Theorem 3.9 *Given a seed set S, our greedy partitioning strategy in Algorithm 2 guarantees an expected revenue to the host which is at least $\frac{1}{2}$ of the maximum expected revenue that could be obtained from an optimal partitioning of the seed set S.*

Proof Let us denote by $C(u)$ the campaigner to which a seed node u is assigned to by following our greedy algorithm. Also, we denote by $C^*(u)$ the campaigner to which a seed node u would have been assigned to in the optimal partitioning of the seed set (Problem 3). If we have a seed node u_{i_1} with $C(u_{i_1}) \neq C^*(u_{i_j})$, then we can form a ring of seed nodes, such that for each seed node u_{i_j}, it holds that $C(u_{i_j}) \neq C^*(u_{i_j})$, and the following is also satisfied.

$$
\begin{aligned}
C^*(u_{i_1}) &= C(u_{i_2}) \\
C^*(u_{i_2}) &= C(u_{i_3}) \\
C^*(u_{i_3}) &= C(u_{i_4}) \\
&\vdots \\
C^*(u_{i_{km-1}}) &= C(u_{i_{km}}) \\
C^*(u_{i_{km}}) &= C(u_{i_1})
\end{aligned} \tag{3.18}
$$

It is always possible to build such a ring of seed nodes because both the greedy and the optimal assignments follow the constraints on the seed set sizes for the client campaigners.

Now, we consider a pair of seed nodes u_{i_j} and $u_{i_{j+1}}$ such that $C^*(u_{i_j}) = C(u_{i_{j+1}})$. There can be two distinct cases based on in which order our greedy algorithm assigns these two seed nodes to the respective campaigners.

Case 1. u_{i_j} is assigned before $u_{i_{j+1}}$. Then, it follows:
$\mathcal{R}_{C(u_{i_j})}(u_{i_j}) \geq \mathcal{R}_{C(u_{i_{j+1}})}(u_{i_j}) = \mathcal{R}_{C^*(u_{i_j})}(u_{i_j})$.

Case 2. $u_{i_{j+1}}$ is assigned before u_{i_j}. Then, it follows:
$\mathcal{R}_{C(u_{i_{j+1}})}(u_{i_{j+1}}) \geq \mathcal{R}_{C(u_{i_{j+1}})}(u_{i_j}) = \mathcal{R}_{C^*(u_{i_j})}(u_{i_j})$.

Since, the greedy algorithm always makes one of the above choices, we get the following inequality.

$$
\mathcal{R}_{C(u_{i_j})}(u_{i_j}) + \mathcal{R}_{C(u_{i_{j+1}})}(u_{i_{j+1}}) \geq \mathcal{R}_{C^*(u_{i_j})}(u_{i_j}) \tag{3.19}
$$

By summing Equation 3.19 over $j = 1$ to km, we get:

$$
2 \times \big[\sum_{j=1}^{km} \mathcal{R}_{C(u_{i_j})}(u_{i_j})\big] \geq \sum_{j=1}^{km} \mathcal{R}_{C^*(u_{i_j})}(u_{i_j}) \tag{3.20}
$$

Hence the theorem. ∎

Corollary 3.10 *Our iterative hill-climbing-based solution of the optimistic seed set selection problem (Problem 2), coupled with the* greedy *partition of those selected seed nodes (Problem 3), generates a solution with approximation guarantee* $\frac{1}{2m}(1 - \frac{1}{e})$ *to the optimal solution of the host's revenue maximization problem (Problem 1) under the* K-LT *model with m campaigners, assuming that each campaigner has the same number of seed nodes.* ◇

Proof The corollary directly follows from Theorem 3.8 and Theorem 3.9. ∎

Time Complexity. We note that there are km seed nodes and each seed node could be repeatedly inserted into $\mathbb{L}$ for at most m times. Therefore, the complexity of repeated insertions into $\mathbb{L}$ is $\mathcal{O}(km^2 \log(km))$. Also, for each insertion of node u into $\mathbb{L}$, we need to find the maximum value in any dimension of the individual revenue vector $\mathcal{R}(u)$. Assuming that the dimensions of $\mathcal{R}(u)$ are pre-sorted based on their values, we can find the maximum value in any dimension of $\mathcal{R}(u)$ in the time $\mathcal{O}(\log m)$. Therefore, the total time complexity of at most km^2 insertions into $\mathbb{L}$ is given by $\mathcal{O}(km^2 \log(km) + km^2 \log m)$, which is $\mathcal{O}(km^2 \log(km))$. In addition, we require $\mathcal{O}(kmn + ekm)$ time to compute the individual revenue vectors for all seed nodes. Therefore, the overall time complexity of our greedy approach is $\mathcal{O}(kmn + ekm + km^2 \log(km))$.

3.6.5 Exact Solution for Directed Trees

In this section, we show that the revenue maximization problem under the K-LT model can be solved optimally in polynomial time over a tree dataset. Our proof is based on a reduction of the current problem into an equivalent problem of revenue maximization under the MCIC model over a tree dataset, which can be solved optimally in polynomial time as illustrated earlier in Section 3.5.

Our reduction works as follows. Since each node in a tree has at most one incoming edge, we eliminate those incoming edges for which the probability on the edge is less than the activation threshold of the destination node. On the other hand, we retain those incoming edges for which the edge-probability is higher than or equal to the activation threshold of the destination node, and we reassign a probability 1 to all these retained edges. One may note that all the retained edges are also independent to each other even under the K-LT model, as each node has at most one retained incoming edge in a tree. Therefore, we can apply the dynamic-programming-based exact solution for the MCIC model to find an exact solution in polynomial time in the size of the tree dataset.

The above reduction only works if the thresholds of the nodes are fixed. If the thresholds are chosen randomly, we can assign the probability of activating the node, if the parent is activated, to the incoming edge. For the same reasons as above, we can apply our dynamic-programming-based exact solution for the MCIC model on the adjusted tree to find an exact solution in polynomial time in the size of the tree dataset.

3.7 Possible Extensions

In this section we want to explore two possible extensions to our Problem 1. The **first** possible extension is motivated by the possibility that other campaigners, which do not directly provide a product, do want as few product adaptions as

possible. For example if multiple car companies want to run campaigns for their cars, then environmentalists, the politics and the public transportation system might be interested, that as few adaptions as possible are made, independent of the specific end product. Another motivation might be to find *k-Effectors* [7] for multiple campaigners.

The first extension adds revenue for all nodes, which remain uninfluenced until the end. For convenience we denote all uninfluenced nodes to be influenced by C_0's campaign. This extends the *revenue matrix* from $(A_{iu})_{m\times n}$ to $(A_{iu})_{(m+1)\times n}$. We also denote by $Pr(u, 0, S)$ the probability that a node remains uninfluenced until the end. We then only need to adjust Problem 1 to start iterating from $i = 0$ instead of $i = 1$.

We can adjust our algorithm for the MCIC model described in Section 3.5.2 by replacing $A_{j,v} \times \mathcal{B}(u, v)$ with $A_{j,v} \times \mathcal{B}(u, v) + A_{0,v} \times (1 - \mathcal{B}(u, v))$ in F_1 of the OPT function, which adds the revenue for remaining uninfluenced multiplied by the probability to remain uninfluenced. Unfortunately the extended problem is neither monotonic nor submodular if adapted to one campaigner as described in Section 3.4, therefore the baseline algorithms and the first step of our solution for Problem 2 do not provide any performance guarantees. On the other side, our solution for Problem 3 still works without any change, because the activation of nodes is independent of the assigned campaigners.

The **second** possible extension is motivated by secrets, which should not reach certain persons. For example if a company wants to develop a new product, which persons should they include in the development team, if they want to keep it secret. Or if they want to sell a brand product much cheaper under a different brand to gain additional sales, in a different market segment, which cares more about the price, then they do not want to lose sales in the original market segment. That means that persons who probably buy the product under the more expensive brand, should not be influenced by the campaign for the cheaper product.

The second extension allows the values in the *revenue matrix* to be negative as well as positive. The problems of this extension are very similar to the ones of the first extension, as again the extended problem is neither monotonic nor submodular if adapted to one campaigner as described in Section 3.4 and therefore our baseline algorithms and the first step of our solution under the K-LT model do not provide any performance guarantees. On the other hand our solution under the MCIC model and the second step of our solution to the K-LT model still work.

Chapter 4

Experiments

In this chapter we present experimental results which illustrate the effectiveness, efficiency, and scalability of our revenue maximization algorithms. The code is implemented in C++ and the experiments were performed on a single core of a 132GB, 2.26GHz Xeon server.

4.1 Datasets

We summarize our data sets in Table 4.1.

Table 4.1: Dataset Characteristics

Dataset	# Nodes	# Edges	Edge Prob: Mean, SD, Quartiles
Flickr	78 322	20 343 018	0.09 ± 0.06, $\{0.06, 0.07, 0.09\}$
DBLP	684 911	4 569 982	0.08 ± 0.07, $\{0.05, 0.05, 0.10\}$
NetHEPT	15 229	62 752	0.28 ± 0.28, $\{0.0006, 0.27, 0.53\}$
NetHEPT-Tree	15 229	13 452	0.25 ± 0.28, $\{0.0005, 0.21, 0.51\}$

Flickr (http://www.flickr.com). Flickr is an online community, where users share photos, and participate in common-interest groups. We borrowed the dataset from [26], where the probability of an edge between any two users is computed assuming *homophily*; in particular, the Jaccard coefficient of the interest groups that the two users belong to.

DBLP (http://www.informatik.uni-trier.de/~ley/db/). The dataset is a subset of the popular co-authorship network used in [26]. Here, the edge probabilities express the strength of the collaboration between the two incident authors. Particularly, in [26], the probabilities derive from an exponential cumulative distribution function to the number of collaborations; hence, if two authors collaborated c times, we assign the corresponding probability as $1 - \exp^{-c/10}$.

NetHEPT (http://www.arXiv.org). This graph is created from the "High Energy Physics - Theory" section of the arXiv with papers from 1991 to 2003 [12]. Since there are no edge probabilities in this graph dataset, we synthetically assign probabilities to the edges that simulate the community structure in a social network. We identify 60 non-overlapping communities from this graph dataset, each with 170 nodes. If an edge is completely inside a cluster, we uniformly assign a probability between 0.2 to 0.8; all other edges are assigned probabilities uniformly from 0 to 0.001.

NetHEPT-Tree. In order to verify the performance of our methods over a tree dataset, we also consider a spanning-tree of the original *NetHEPT* graph. The spanning tree is built by first randomly selecting a node and then identifying its breadth-first-search tree.

In all our graph datasets, the edges are directed. In addition, for the K-LT model, in all datasets, if the sum of probabilities of all incoming edges to a node is more than 1, we normalize those edge probabilities by their aggregate value, such that the sum of probabilities for in-edges to every node is no more than 1 [26].

4.2 Number of Campaigners and Seed Nodes

We vary the number of campaigners from 2 to 5, while the number of seed nodes per campaigner is varied from 5 to 30.

4.3 Revenue Distributions

We consider three types of revenue distribution in order to simulate various real-world scenarios with either a **Revenue Minimum** (RMin) of 0.0 or 0.1 for each revenue-matrix-element $A_{i,u}$.

Uniform (U). In this setting, each campaigner selects its target users uniformly over the network and independent of other campaigners. Therefore, in our framework, we assign every revenue-matrix-element $A_{i,u} = 1$ monetary unit (RMax) with probability $\frac{1}{m}$; and $A_{i,u} =$ RMin monetary unit with probability $(1 - \frac{1}{m})$. Here, m is the number of campaigners. One may note that we have normalized the amount of money that a campaigner gives to the host for one user on a scale from RMin to 1 monetary unit.

Clustering with Low Competition (CLC). In this setting, we assume that each campaigner's target users form certain clusters in the network. In addition, we also assume that there are some users who belong to target sets of all the campaigners. We call this model "clustering with low competition" as we limit

the ratio of such mutually overlapping target users to a relatively small percentage. We simulate this setting as follows. We first partition the graph into 15 non-overlapping and highly-connected clusters, each cluster having an equal number of nodes. For the first 5 clusters, all $A_{i,u}$ values are set to 1 monetary unit (RMax), i.e., 33% of the nodes belong to the target users of all campaigners. The remaining clusters are assigned to the campaigners in a round-robin manner. If a cluster is assigned to the campaigner C_j as its target set, then we assign each $A_{j,u} = 0.5$ monetary units (RMid), and the remaining $A_{i,u} =$ RMin monetary units, for all $j \neq i$, inside that cluster.

Clustering with High Competition (CHC). This setting is similar to the previous CLC setting — the only difference is that there is a relatively large number of users who belong to the target sets of all campaigners. We simulate this setting as before; however, we assign the first 10 out of 15 clusters as the target sets for all campaigners. This implies that 67% of the nodes belong to the target users of all campaigners.

4.4 Comparing Methods

We compare the following techniques.

Random. We randomly select a distinct seed set for each campaigner. In our experiments, we did 10 runs of the Random method, and selected the one with the maximum revenue out of all these 10 runs.

RevMax-Separate (RevMax-S). This are our baseline methods described in Section 3.5.1 and 3.6.1, which consider *one campaigner at a time* while selecting the seed sets.

RevMax-Combined (RevMax-C). This are our proposed dynamic programming methods described in Sections 3.5 and 3.6 for the revenue maximization problem, where we indeed consider the *competition among multiple campaigners* while selecting the seed sets for all the campaigners.

RevMax-Combined-Greedy (RevMax-C-G). This is our greedy method described in Sections 3.6.4 for the K-LT information diffusion model. This method also considers the *competition among multiple campaigners* while selecting the seed sets for all the campaigners.

We compare the four aforementioned techniques under both the MCIC and the K-LT models, except for the RevMax-C-G method, which we only compare under the K-LT model, because this algorithm does not work under the MCIC model. As the underlying viral marketing method in RevMax-S, RevMax-C and RevMax-C-G, we use the CELF++ algorithm [9, 10] due to its efficiency.

4.5 Evaluation Metrics

We compare the host's revenues obtained from RevMax-S, RevMax-C and RevMax-C-G techniques with that of the Random method. For this purpose, we design the following metric.

Revenue Improvement Rate (RIR). This is defined as the ratio of the host's expected revenue obtained from the seed sets as identified by RevMax-C (or by RevMax-S/RevMax-C-G) with respect to the host's expected revenue obtained from a random selection of seed sets (Random).

Finally, we also compare the efficiency of identifying the seed sets using the three different methods RevMax-S, RevMax-C and RevMax-C-G.

We apply the Monte-Carlo sampling over the entire graph (with 1 000 samples [26]) to compute the host's expected revenue. For some experiments we limit the calculations of the expected spread of information of CELF++ to 1 or 2 rounds as described in Section 3.2, due to the huge calculation time CELF++ would need otherwise. For the same reason we also reduce the number of samples in CELF++ from 1 000 to 200 for some experiments, but we never reduce the number of samples or the number of rounds for the final revenue results of the seed sets.

Chapter 5

Results

We first show our results under the MCIC model in Section 5.1, and then under the K-LT model in Section 5.2. At the end of this chapter we show our results on scalability in Section 5.3. We discuss the results in the next chapter.

5.1 Performance with the MCIC Model

In Table 5.1 and 5.2 are the results on the Revenue Improvement Rate for the *NetHEPT* dataset under the MCIC information diffusion model with 2 campaigners and a RMin of 0.0 or 0.1. In Table 5.3 we can see results with a third campaigner. We can see that in most cases our dynamic programming algorithm RevMax-C outperforms the baseline algorithm RevMax-S. Figure 5.1 shows the seed sets finding times with 2 campaigners. The results show that the CELF++ algorithm with infinite rounds has huge seed set sets finding times, which is the reason, why we limited the number of rounds to 1 or 2 in most experiments under the MCIC model.

In Table 5.4 are the results for the three datasets which we use. Due to the size of some of the bigger datasets and the time complexity of our dynamic programming algorithm RevMax-C, we only run experiments with 2 campaigners and 5 seeds per campaigner. The results show that in most cases our dynamic programming algorithm RevMax-C has an equal or better performance than the baseline algorithm RevMax-S. The corresponding results for the seed sets finding times are in Figure 5.2, which shows that for the *NetHEPT* dataset both algorithms need about the same time, whereas for the *DBLP* and the *Flickr* datasets our dynamic programming algorithm RevMax-C needs more computation time by a factor of about 30 and 3, respectively, than our baseline algorithm RevMax-S. However we only calculated 1 round in the baseline algorithm for those two datasets, due to the seed sets finding times being too long with more rounds to finish in reasonable time.

In Table 5.5 and Figure 5.3 are the results for the *NetHEPT-Tree* dataset. As our dynamic programming algorithm RevMax-C computes an optimal solution

on trees, we are always better than the baseline algorithm RevMax-S. The seed sets finding times for RevMax-C on the tree are very similar to the seed sets finding times on the whole graph, as we expected. On the other side the baseline algorithm RevMax-S is much faster on the tree, due to the much lower information propagation of the tree, which reduces the time to calculate the expected revenue for each node.

The Host's Expected Revenue for the *NetHEPT* dataset under the MCIC information diffusion model is shown in Table 5.6. We present the results for two different sets of revenue distribution values. The values of the **Revenue Maximum** (RMax) and the **Revenue Middle** (RMid) are ten times higher in the second case than in the first one. The results show that in general the Host's Expected Revenue is also about ten times higher in the second case compared to the first case. This shows that scaling the revenue distribution values linear also scales the Host's Expected Revenue by the same multiplier.

In Table 5.7 are the results for 2 campaigners with different numbers of seed nodes on the *NetHEPT* dataset under the MCIC information diffusion model with the CHC revenue distribution. The results show that our dynamic programming algorithm RevMax-C is better than the baseline algorithm RevMax-S independent of the calculation order of the 2 campaigners.

We present in Table 5.8 the results of our experiments with different Revenue Distribution Values for the 2 campaigners. The campaigner C_1 has RMin $= 0$, RMid $= 6$ and RMax $= 10$. The campaigner C_2 has RMin $= 0$, RMid $= 3$ and RMax $= 10$. Additionally those experiments have an adjusted Revenue Distribution. The CHC 9 and CLC 3 revenue distributions have 9 and 3 clusters as target sets for all campaigners instead of 10 and 5, respectively. Of the remaining clusters are twice as many assigned to the campaigner C_2 than to the campaigner C_1, so that the sum of the revenue values of all nodes is about the same for both campaigners. The results show that our dynamic programming algorithm RevMax-C is better than the baseline algorithm RevMax-S independent of the calculation order of the 2 campaigners, except for one case.

5.2 Performance with the K-LT Model

In Table 5.9 and 5.10 are the results on the Revenue Improvement Rate for the *NetHEPT* dataset under the K-LT information diffusion model with 2 campaigners and a RMin of 0.0 or 0.1. The performance of the baseline algorithm RevMax-S is about as many times worse as it is better than the performance of the dynamic programming algorithm RevMax-C. The performance of the greedy algorithm RevMax-C-G is always very close to the performance of the dynamic programming algorithm RevMax-C.

In Table 5.11 and 5.12 we can see the results for more campaigners on the

Table 5.1: Revenue Improvement Rate (RIR), MCIC Information Diffusion Model, 2 Campaigners, *NetHEPT* Dataset, RMin = 0.0, CELF++: 2 rounds, 1000 samples

Revenue Distribution	#Seed Nodes per Camp.	RIR RevMax-S	RIR RevMax-C
CHC	5	2.50	**3.13**
	10	3.04	**3.31**
	15	2.20	**2.49**
	20	1.60	**1.84**
CLC	5	**3.31**	3.05
	10	1.72	**1.89**
	15	2.59	**3.09**
	20	2.98	**3.24**
U	5	3.22	**3.63**
	10	**2.01**	1.92
	15	3.62	**3.93**
	20	2.13	**2.23**

Table 5.2: Revenue Improvement Rate (RIR), MCIC Information Diffusion Model, 2 Campaigners, *NetHEPT* Dataset, RMin = 0.1, CELF++: 2 rounds, 1000 samples

Revenue Distribution	#Seed Nodes per Camp.	RIR RevMax-S	RIR RevMax-C
CHC	5	2.52	**3.14**
	10	2.92	**3.28**
	15	2.68	**3.07**
	20	1.94	**2.23**
CLC	5	3.30	**3.33**
	10	2.91	**3.20**
	15	2.48	**2.94**
	20	2.09	**2.37**
U	5	3.23	**3.52**
	10	**2.12**	2.04
	15	2.72	**2.80**
	20	2.34	**2.52**

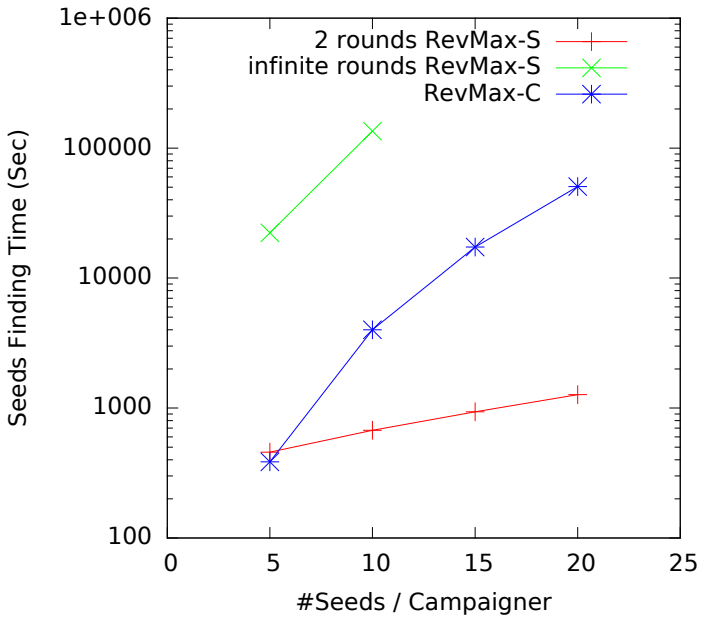


Figure 5.1: Seed Sets Finding Time, MCIC Information Diffusion Model, 2 Campaigners, *NetHEPT* Dataset, CELF++: 1000 samples

Table 5.3: Revenue Improvement Rate (RIR), MCIC Information Diffusion Model, 5 Seeds/Campaigner, *NetHEPT* Dataset, RMin = 0.1, CELF++: infinite rounds, 1000 samples

Revenue Distribution	#Camp.	RIR RevMax-S	RIR RevMax-C
CLC	2	2.98	**3.23**
	3	2.36	**2.84**
U	2	3.31	**3.50**
	3	2.31	**3.59**

Table 5.4: Revenue Improvement Rate (RIR), MCIC Information Diffusion Model, 2 Campaigners, 5 Seeds/Campaigner, RMin = 0.1

Dataset	Revenue Distribution	RIR RevMax-S	RIR RevMax-C
NetHEPT	CHC	2.52	**3.14**
	CLC	3.30	**3.33**
	U	3.23	**3.52**
DBLP	CHC	**1.02**	**1.02**
	CLC	1.02	**1.03**
	U	**1.03**	**1.03**
Flickr	CHC	1.43	**1.66**
	CLC	**1.20**	1.15
	U	1.01	**1.11**

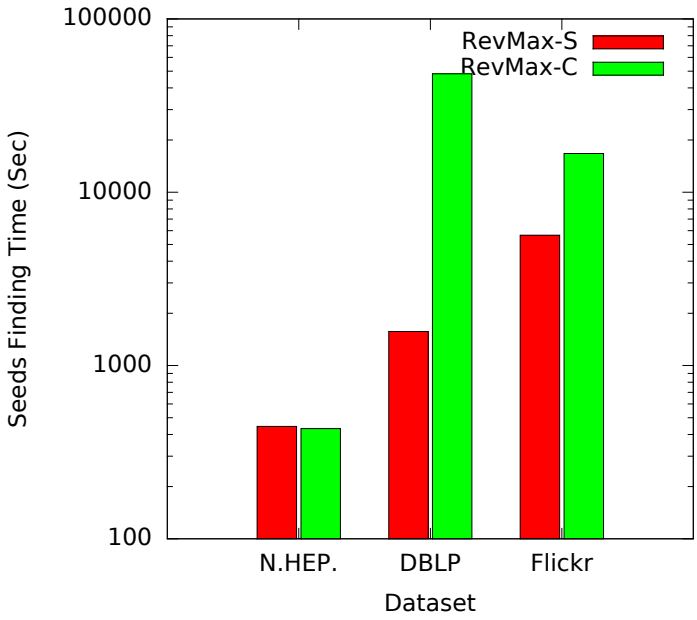


Figure 5.2: Seed Sets Finding Time, MCIC Information Diffusion Model, 2 Campaigners, 5 Seeds/Campaigner

Table 5.5: Revenue Improvement Rate (RIR), MCIC Information Diffusion Model, 2 Campaigners, *NetHEPT-Tree* Dataset, RMin = 0.0, CELF++: infinite rounds, 1000 samples

Revenue Distribution	#Seed Nodes per Camp.	RIR RevMax-S	RIR RevMax-C
CHC	5	13.46	**14.17**
	10	9.44	**10.30**
	15	11.10	**11.86**
	20	9.32	**9.94**
CLC	5	15.97	**16.77**
	10	10.03	**10.48**
	15	8.66	**9.50**
	20	10.71	**11.52**
U	5	9.33	**9.82**
	10	8.98	**9.58**
	15	8.87	**9.44**
	20	8.26	**8.83**

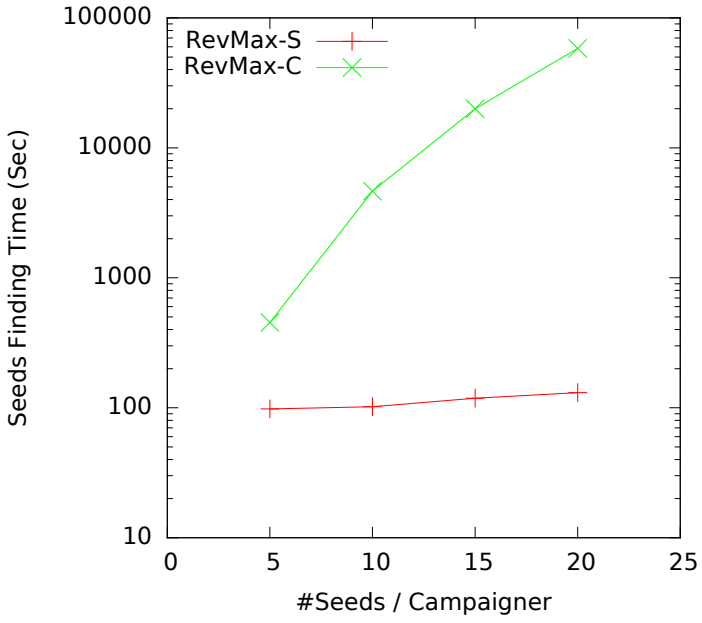


Figure 5.3: Seed Sets Finding Time, MCIC Information Diffusion Model, 2 Campaigners, *NetHEPT-Tree* Dataset, CELF++: infinite rounds, 1000 samples

Table 5.6: Host's Expected Revenue (HER), MCIC Information Diffusion Model, 2 Campaigners, *NetHEPT* Dataset, CLC Revenue Distribution, CELF++: 2 rounds, 1000 samples

Revenue Distribution Values	#Seed Nodes per Camp.	HER RevMax-S	HER RevMax-C
{RMin = 0.1, RMid = 0.5, RMax = 1.0}	5	665.6	**672.2**
	10	873.8	**962.6**
	15	1032.9	**1222.7**
	20	1202.8	**1363.0**
{RMin = 0.0, RMid = 5.0, RMax = 10.0}	5	**6452**	5934
	10	8476	**9336**
	15	9990	**11903**
	20	12313	**13355**

Table 5.7: Revenue Improvement Rate (RIR), MCIC Information Diffusion Model, 2 Campaigners, *NetHEPT* Dataset, CHC Revenue Distribution, RMin = 0.0, CELF++: 2 rounds, 1000 samples

#Seed Nodes for Camp. C_1	#Seed Nodes for Camp. C_2	RIR RevMax-S Order: C_1, C_2	RIR RevMax-S Order: C_2, C_1	RIR RevMax-C
5	10	2.32	2.32	**2.61**
10	20	2.08	2.09	**2.53**

Table 5.8: Revenue Improvement Rate (RIR), MCIC Information Diffusion Model, 2 Campaigners, *NetHEPT* Dataset, C_1 : {RMin = 0, RMid = 6, RMax = 10}, C_2 : {RMin = 0, RMid = 3, RMax = 10}, CELF++: 2 rounds, 1000 samples

Revenue Distribution	#Seed Nodes per Camp.	RIR RevMax-S Order: C_1, C_2	RIR RevMax-S Order: C_2, C_1	RIR RevMax-C
CHC 9	5	2.65	2.66	**3.05**
	10	3.35	3.57	**4.22**
	15	1.78	1.77	**2.09**
	20	2.48	2.39	**2.73**
CLC 3	5	**3.01**	2.69	2.96
	10	2.97	3.18	**3.60**
	15	2.35	2.35	**2.47**
	20	1.86	1.77	**1.96**

NetHEPT dataset under the K-LT information diffusion model with the **Clustering with High Competition** (CHC) revenue distribution. The results show that our dynamic programming algorithm RevMax-C has better performance with lower numbers of seed nodes per campaigner and worse performance with higher numbers of seed nodes per campaigner than our baseline algorithm RevMax-S. Again the performance of our greedy algorithm RevMax-C-G is very close to the one of RevMax-C.

In Figure 5.4, 5.5 and 5.6 are the results of the seed sets finding times for 2, 3 and 5 campaigners on the *NetHEPT* dataset under the K-LT information diffusion model. The seed sets finding times for the dynamic programming algorithm RevMax-C are almost always exactly the same as the ones for the greedy algorithm RevMax-C-G. The only exceptions are with higher numbers of seed nodes per campaigner and 5 campaigners, but even under those conditions the difference is below 0.2%. The reason is that the size of Problem 2 (15'229 nodes) is much larger than the size of Problem 3 ($5 \times 30 = 150$ seed nodes), therefore the computation of the seed set partitioning does influence the overall seed sets finding times very little. The results in the figures show that the seed sets finding times of our dynamic programming algorithm RevMax-C and our greedy algorithm RevMax-C-G increase faster than the ones for the baseline algorithm RevMax-S.

In Table 5.13 are the results on the Revenue Improvement Rate for the *DBLP* dataset under the K-LT information diffusion model with 2, 3 and 5 campaigners and a RMin of 0.1. The revenue distribution is **Clustering with Low Competition** (CLC). The results show that the dynamic programming algorithm

RevMax-C preforms better with lower numbers of seed nodes and worse with higher numbers of seed nodes compared to the baseline algorithm RevMax-S. The corresponding seed sets finding times are presented in Figure 5.7 and show that for lower numbers of seed nodes per campaigner our dynamic programming algorithm RevMax-C and our greedy algorithm RevMax-C-G are faster than the baseline algorithm RevMax-S, but the seed sets finding times increase faster with increasing numbers of seed nodes per campaigner.

In Table 5.14 we see the results on the Revenue Improvement Rate for the *Flickr* dataset under the K-LT information diffusion model with 2, 3 and 5 campaigners, 5 seed nodes per campaigner and a RMin of 0.0. Under those conditions our dynamic programming algorithm RevMax-C performs better in most cases than our baseline algorithm RevMax-S. Again the performance of our greedy algorithm RevMax-C-G is very close to the performance of RevMax-C. In Figure 5.8 are the results of the corresponding seed sets finding times, which show that RevMax-S, RevMax-C and RevMax-C-G scale similar in seed sets finding time with an increasing number of campaigners.

The Host's Expected Revenue for the *Flickr* dataset under the K-LT information diffusion model is shown in Table 5.15. We present the results for two different sets of revenue distribution values. The values of the **Revenue Maximum** (RMax) and the **Revenue Middle** (RMid) are ten times higher in the second case than in the first one. The results show that in general the Host's Expected Revenue is also about ten times higher in the second case compared to the first case. This shows that scaling the revenue distribution values linear also scales the Host's Expected Revenue by the same multiplier.

In Table 5.16 are the results for 2 campaigners with different numbers of seed nodes on the *NetHEPT* dataset under the K-LT information diffusion model with the CHC revenue distribution. The results show that our dynamic programming algorithm RevMax-C is worse than the baseline algorithm RevMax-S independent of the calculation order of the 2 campaigners.

We present in Table 5.17 the results of our experiments with different Revenue Distribution Values for the 2 campaigners. The campaigner C_1 has RMin $= 0$, RMid $= 6$ and RMax $= 10$. The campaigner C_2 has RMin $= 0$, RMid $= 3$ and RMax $= 10$. Additionally those experiments have an adjusted Revenue Distribution. The CHC 9 and CLC 3 revenue distributions have 9 and 3 clusters as target sets for all campaigners instead of 10 and 5, respectively. Of the remaining clusters are twice as many assigned to the campaigner C_2 than to the campaigner C_1, so that the sum of the revenue values of all nodes is about the same for both campaigners. The results show that our dynamic programming algorithm RevMax-C is worse than the baseline algorithm RevMax-S independent of the calculation order of the 2 campaigners, except for three cases.

Table 5.9: Revenue Improvement Rate (RIR), K-LT Information Diffusion Model, 2 Campaigners, *NetHEPT* Dataset, RMin = 0.0, CELF++: infinite rounds, 1000 samples

Revenue Distribution	#Seed Nodes per Camp.	RIR RevMax-S	RIR RevMax-C	RIR RevMax-C-G
CHC	5	**9.20**	9.16	9.16
	10	7.82	**7.98**	**7.98**
	15	**6.93**	6.88	6.85
	20	**6.02**	5.52	5.50
	25	**4.76**	4.47	4.44
	30	5.32	**5.34**	**5.34**
CLC	5	8.52	**8.83**	8.64
	10	**9.88**	9.71	9.59
	15	**7.42**	7.28	7.16
	20	**6.69**	6.66	6.56
	25	**6.12**	5.99	5.96
	30	**6.36**	6.07	6.06
U	5	**5.25**	5.15	5.04
	10	7.69	**8.38**	8.34
	15	5.10	**5.15**	5.13
	20	5.92	**5.99**	**5.99**
	25	5.09	**5.19**	**5.19**
	30	4.32	**4.36**	4.34

Table 5.10: Revenue Improvement Rate (RIR), K-LT Information Diffusion Model, 2 Campaigners, *NetHEPT* Dataset, RMin = 0.1, CELF++: infinite rounds, 1000 samples

Revenue Distribution	#Seed Nodes per Camp.	RIR RevMax-S	RIR RevMax-C	RIR RevMax-C-G
CHC	5	8.99	**9.20**	**9.20**
	10	7.73	**7.97**	**7.97**
	15	**6.89**	6.82	6.79
	20	**6.00**	5.59	5.56
	25	**4.65**	4.54	4.52
	30	**5.65**	5.53	5.51
CLC	5	8.62	**8.67**	8.52
	10	9.52	**9.53**	9.42
	15	**7.06**	6.98	6.93
	20	6.40	**6.45**	6.43
	25	**5.80**	5.78	5.60
	30	**6.08**	5.82	5.74
U	5	5.06	**5.29**	5.26
	10	7.93	**8.38**	8.35
	15	5.12	**5.17**	5.16
	20	5.96	**6.04**	6.01
	25	**5.10**	5.05	5.02
	30	**4.23**	4.06	4.05

Table 5.11: Revenue Improvement Rate (RIR), K-LT Information Diffusion Model, *NetHEPT* Dataset, CHC Revenue Distribution, RMin = 0.0, CELF++: infinite rounds, 1000 samples

#Camp.	#Seed Nodes per Camp.	RIR RevMax-S	RIR RevMax-C	RIR RevMax-C-G
2	5	**9.20**	9.16	9.16
	10	7.82	**7.98**	**7.98**
	15	**6.93**	6.88	6.85
	20	**6.02**	5.52	5.50
	25	**4.76**	4.47	4.44
	30	5.32	**5.34**	**5.34**
3	5	5.78	**5.84**	5.83
	10	7.09	**7.42**	7.33
	15	**6.28**	6.26	6.17
	20	5.95	**6.03**	5.98
	25	**5.93**	5.75	5.67
	30	**5.09**	4.56	4.52
5	5	5.84	**6.04**	5.97
	10	**5.15**	5.00	4.92
	15	**5.24**	4.86	4.79
	20	**5.07**	4.74	4.65
	25	**4.73**	4.12	4.02
	30	**4.37**	3.92	3.87

Table 5.12: Revenue Improvement Rate (RIR), K-LT Information Diffusion Model, *NetHEPT* Dataset, CHC Revenue Distribution, RMin = 0.1, CELF++: infinite rounds, 1000 samples

#Camp.	#Seed Nodes per Camp.	RIR RevMax-S	RIR RevMax-C	RIR RevMax-C-G
2	5	8.99	**9.20**	**9.20**
	10	7.73	**7.97**	**7.97**
	15	**6.89**	6.82	6.79
	20	**6.00**	5.59	5.56
	25	**4.65**	4.54	4.52
	30	**5.65**	5.53	5.51
3	5	5.86	**5.94**	5.88
	10	7.01	**7.29**	7.26
	15	**6.10**	5.73	5.63
	20	**5.87**	5.73	5.70
	25	**5.81**	5.56	5.52
	30	**4.89**	4.36	4.33
5	5	5.70	**5.85**	5.77
	10	**5.00**	4.83	4.79
	15	**5.04**	4.77	7.72
	20	**4.91**	4.65	4.57
	25	**4.58**	3.08	3.02
	30	**4.24**	3.80	3.74

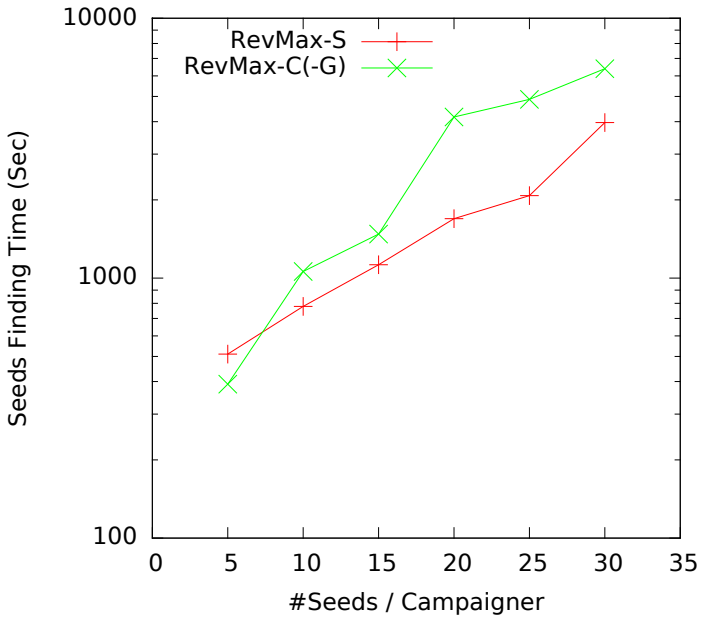


Figure 5.4: Seed Sets Finding Time, K-LT Information Diffusion Model, 2 Campaigners, *NetHEPT* Dataset, CELF++: infinite rounds, 1000 samples

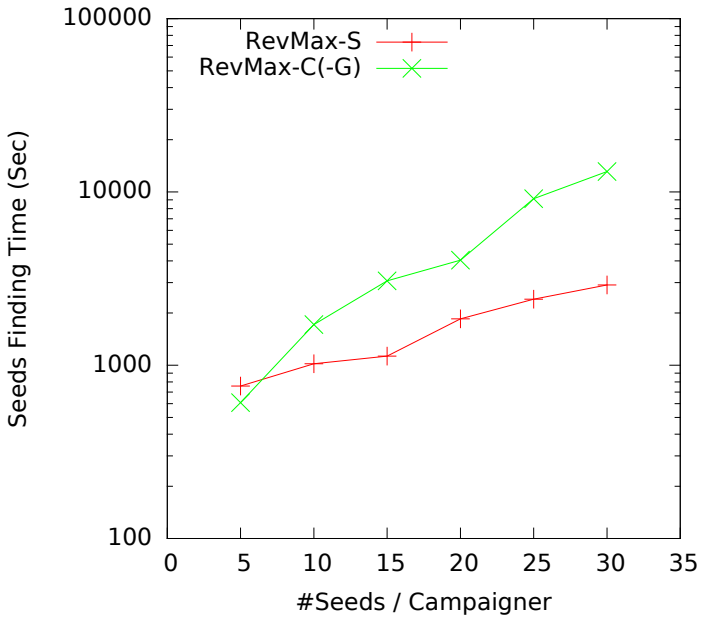


Figure 5.5: Seed Sets Finding Time, K-LT Information Diffusion Model, 3 Campaigners, *NetHEPT* Dataset, CELF++: infinite rounds, 1000 samples

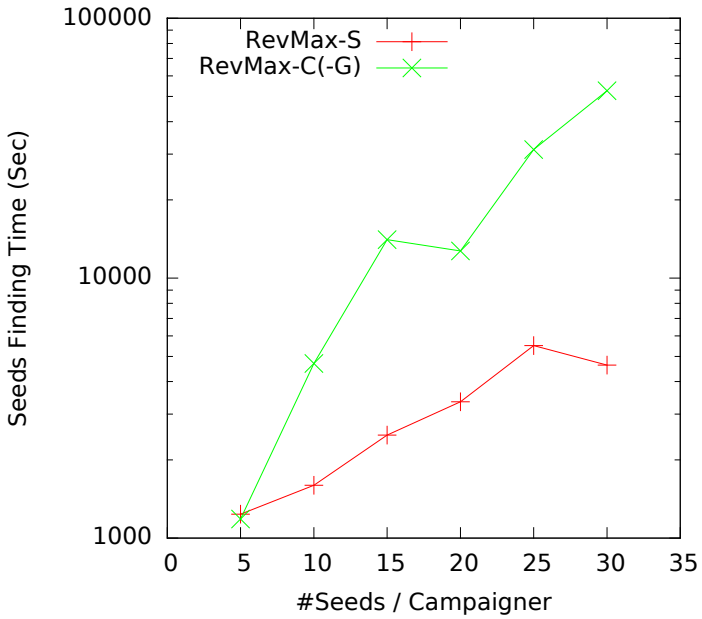


Figure 5.6: Seed Sets Finding Time, K-LT Information Diffusion Model, 5 Campaigners, *NetHEPT* Dataset, CELF++: infinite rounds, 1000 samples

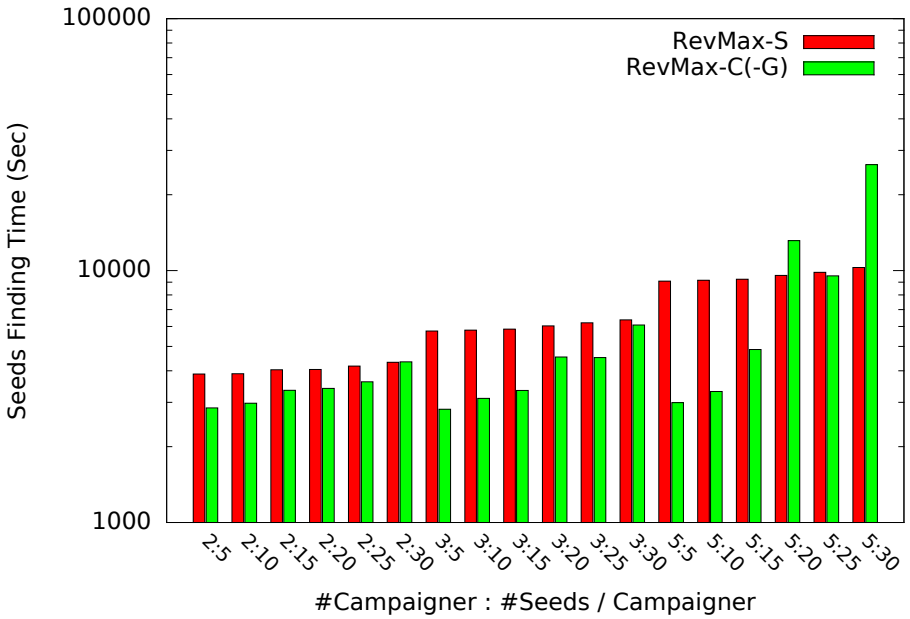


Figure 5.7: Seed Sets Finding Time, K-LT Information Diffusion Model, *DBLP* Dataset, CELF++: 1 round, 200 samples

Table 5.13: Revenue Improvement Rate (RIR), K-LT Information Diffusion Model, CLC Revenue Distribution, *DBLP* Dataset, RMin = 0.1, CELF++: 1 round, 200 samples

#Camp.	#Seed Nodes per Camp.	RIR RevMax-S	RIR RevMax-C	RIR RevMax-C-G
2	5	**74.97**	73.52	73.50
	10	43.66	**44.64**	44.43
	15	35.78	**36.68**	36.60
	20	**40.27**	39.35	39.28
	25	**30.54**	30.25	30.19
	30	39.98	**40.32**	40.27
3	5	25.87	**25.99**	25.86
	10	34.19	**34.57**	34.37
	15	47.56	**48.62**	48.53
	20	36.12	**36.82**	36.68
	25	33.98	**35.12**	34.85
	30	**34.94**	33.76	33.39
5	5	47.07	**47.40**	47.26
	10	**36.06**	36.03	35.24
	15	**33.44**	33.41	33.23
	20	**32.02**	30.43	30.27
	25	33.72	**33.88**	33.56
	30	**28.47**	27.36	27.05

Table 5.14: Revenue Improvement Rate (RIR), K-LT Information Diffusion Model, 5 Seeds/Campaigner, *Flickr* Dataset, RMin = 0.0, CELF++: 1 round, 200 samples

Revenue Distribution	#Camp.	RIR RevMax-S	RIR RevMax-C	RIR RevMax-C-G
CHC	2	5.76	**5.95**	**5.95**
	3	**7.04**	6.94	6.93
	5	6.54	**6.91**	6.90
CLC	2	5.57	**5.91**	**5.91**
	3	7.03	**7.27**	7.25
	5	6.56	**6.65**	6.64
U	2	12.20	**12.33**	**12.33**
	3	19.27	**19.71**	19.67
	5	**8.94**	7.15	7.13

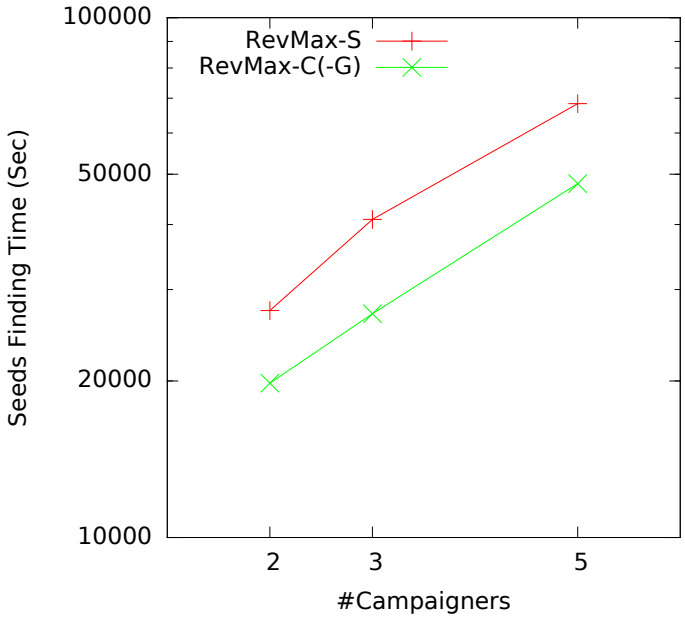


Figure 5.8: Seed Sets Finding Time, K-LT Information Diffusion Model, 5 Seeds/Campaigner, *Flickr* Dataset, CELF++: 1 round, 200 samples

Table 5.15: Host's Expected Revenue (HER), K-LT Information Diffusion Model, 2 Campaigners, 5 Seeds/Campaigner, *Flickr* Dataset, CELF++: 1 round, 200 samples

Revenue Distribution Values	Revenue Distribution	HER RevMax-S	HER RevMax-C	HER RevMax-C-G
{RMin = 0.1,	CHC	5069	**5229**	**5229**
RMid = 0.5,	CLC	3552	**3737**	**3737**
RMax = 1.0}	U	**3637**	3493	3493
{RMin = 0.0,	CHC	53232	**55046**	**55046**
RMid = 5.0,	CLC	35202	**37344**	**37344**
RMax = 10.0}	U	34798	**35173**	35169

Table 5.16: Revenue Improvement Rate (RIR), K-LT Information Diffusion Model, 2 Campaigners, *NetHEPT* Dataset, CHC Revenue Distribution, RMin = 0.0, CELF++: infinite rounds, 1000 samples

#Seed Nodes for Camp. C_1	#Seed Nodes for Camp. C_2	RIR RevMax-S Order: C_1, C_2	RIR RevMax-S Order: C_2, C_1	RIR RevMax-C
5	10	6.00	**6.09**	5.93
10	20	7.36	**7.42**	6.18
15	30	**6.07**	6.03	5.87

Table 5.17: Revenue Improvement Rate (RIR), K-LT Information Diffusion Model, 2 Campaigners, *NetHEPT* Dataset, C_1 : {RMin = 0, RMid = 6, RMax = 10}, C_2 : {RMin = 0, RMid = 3, RMax = 10}, CELF++: infinite rounds, 1000 samples

Revenue Distribution	#Seed Nodes per Camp.	RIR RevMax-S Order: C_1, C_2	RIR RevMax-S Order: C_2, C_1	RIR RevMax-C
CHC 9	5	**9.20**	9.06	8.99
	10	**9.42**	9.17	9.28
	15	7.27	**7.42**	7.26
	20	**6.28**	**6.28**	**6.28**
	25	4.86	**4.93**	4.72
	30	6.35	**6.37**	6.16
CLC 3	5	10.43	10.48	**10.61**
	10	13.33	13.67	**13.78**
	15	7.58	**7.62**	7.20
	20	**8.59**	8.57	8.32
	25	**6.79**	6.72	6.20
	30	**7.30**	**7.30**	6.76

5.3 Scalability

In Figure 5.9 are the results of the seed sets finding times for an increasing number of nodes in the graph from the *Flickr* dataset under the MCIC information diffusion model. We can see that our dynamic programming algorithm RevMax-C scales similar in the number of nodes in the graph as our baseline algorithm RevMax-S with a 1 round limit. Experiments with a higher round limit for our baseline algorithm RevMax-S take too much time, due to the high information diffusion in the *Flickr* dataset.

In Figure 5.10 we can see the results of the seed sets finding times for an increasing number of nodes in the graph from the *Flickr* dataset under the K-LT information diffusion model. Our dynamic programming algorithm RevMax-C scales a little better in the number of nodes in the graph than our baseline algorithm RevMax-S with a 1 round limit.

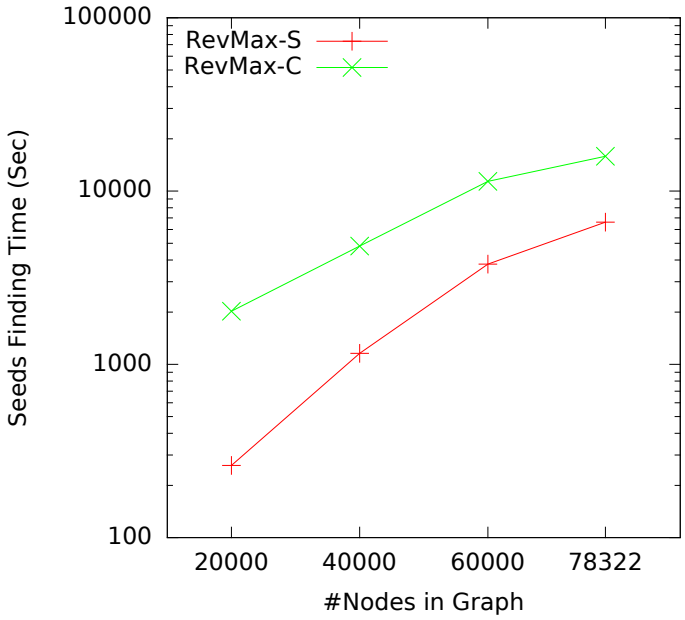


Figure 5.9: Seed Sets Finding Time, MCIC Information Diffusion Model, 2 Campaigners, 5 Seeds/Campaigner, *Flickr* Dataset, CELF++: 1 round, 200 samples

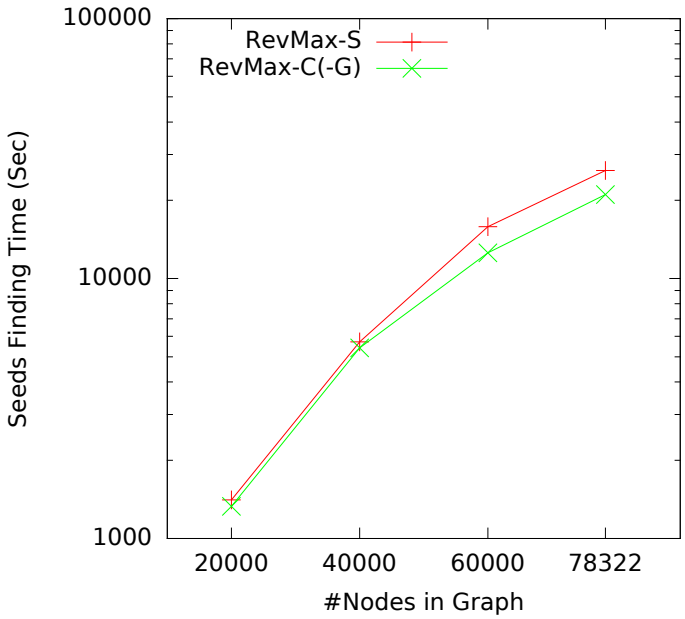


Figure 5.10: Seed Sets Finding Time, K-LT Information Diffusion Model, 2 Campaigners, 5 Seeds/Campaigner, *Flickr* Dataset, CELF++: 1 round, 200 samples

Chapter 6

Discussion

In this chapter we first discuss the results of our experiments, which we presented in the previous chapter and then we make a conclusion.

6.1 Performance with the MCIC Model

The results in Section 5.1 show that our dynamic programming algorithm described in Section 3.5 performs 5% to 10% better than the baseline algorithm in most settings, even with different numbers of seed nodes or different revenue distribution values for the 2 campaigners. In one case the performance is better by over 50% (Table 5.3). But even a 5% to 10% increase in revenue is a significant improvement for the host. Table 5.6 shows that linear scaling does not matter, as we expected of the algorithm's design and the information diffusion model. Therefore the increase in relative revenue is independent of the initial amount.

The results of the baseline algorithm on the *NetHEPT-Tree* dataset are very close to the optimal results of our dynamic programming algorithm, similar to other datasets. This might indicate that the performance of our dynamic programming algorithm is close to the optimal solution on the general graphs, however the performance of the baseline algorithm might be even lower on general graphs compared to the optimal solution, which we do not know.

The worst case scenario for our dynamic programming algorithm would be a graph, where the in-edges of each node have the same probability, as this maximizes the information loss by the extraction of the *most influential tree* described in Section 3.5.4. However such a graph is highly unlikely in social networks, because each person would have to give the same importance to all opinions of all persons they know.

6.2 Performance with the K-LT Model

The results presented in Section 5.2 are closer together than the results for the MCIC information diffusion model. In general our dynamic programming algorithm described in Section 3.6 is better with lower numbers of seed nodes per campaigner than the baseline algorithm, but there are exceptions like the results in Table 5.9. In a lot of cases the results of both algorithms are very close together, the reason might be that both algorithms build on the CELF++ algorithm. As for the MCIC information diffusion model linear scaling also does not matter for our algorithms under the K-LT information diffusion model, as shown in Table 5.15.

Our dynamic programming algorithm for the K-LT information diffusion model performs bad with different numbers of seed nodes or different revenue distribution values for the campaigners compared to the baseline algorithm and in contrast to our dynamic programming algorithm for the MCIC information diffusion model in such settings. The reason might be the optimistic assumption in step 1 of our algorithm (Section 3.6.2), that it is possible to influence each user by a campaign such that the corresponding campaigner spends the maximum amount of money for that particular user. However the settings above make it harder to influence each user according to the assumption.

The results show that the performance of our greedy algorithm described in Section 3.6.4 is very close to the performance of our dynamic programming algorithm, even though the worst case is 50% of the optimum. Nevertheless, the advantage of the lower time complexity of the greedy algorithm is meaningless, because the seed selection step takes a lot more time than the seed partition step of the dynamic programming algorithm or the greedy algorithm. Therefore it is better to use the dynamic programming algorithm in most cases.

6.3 Scalability

Our dynamic programming algorithm for the MCIC information diffusion model scales bad in the number of campaigners and in the number of seed nodes per campaigner, but it scales very good in the number of nodes in the graph compared to the baseline algorithm as shown in Figure 5.9. The results of the baseline algorithm in the figure are with only 1 round and 200 samples. The time the baseline algorithm needs can be multiplied by the number of samples. Also to compute higher numbers of rounds the baseline algorithm needs a lot more time for graphs with high information propagation, whereas our dynamic programming algorithms computation time is uninfluenced by the information propagation characteristics of the graph.

Our algorithms for the K-LT information diffusion model scale very similar, as

all of them are based on the CELF++ algorithm. The differences between the 2 step algorithms, our dynamic programming algorithm and the greedy algorithm, and the baseline algorithm are how often they invoke the CELF++ algorithm and with how many seed nodes as parameter. For example with 3 campaigners and 10 seed nodes per campaigner the baseline algorithm invokes the CELF++ algorithm three times with 10 seed nodes as argument, on the other hand the 2 step algorithms invoke the CELF++ algorithm once with 30 seed nodes as argument. The CELF++ algorithm needs a lot of time to find the first seed node, but it also builds a list of the next best seed nodes during the search. With the help of the list the next seed nodes can be found much faster, but due to the already selected seed nodes, the calculation time of the expected revenue increases with each additional node. This explains the results we see in Figure 5.4, 5.5 and 5.6.

6.4 Conclusion

In this master's thesis, we formulate and investigate the novel problem of revenue maximization of a social network host that sells simultaneous viral marketing campaigns to multiple client campaigners.

While our problem under the IC model and the LT model of information diffusion is **NP**-hard, as well as neither monotonic nor submodular; we develop effective algorithms with theoretical performance guarantees. In addition, our proposed techniques can solve the revenue maximization problem exactly in polynomial time over a tree dataset using both these models.

Our experimental evaluation conducted on various real-world graph datasets and with diverse settings of revenue distributions attest high-quality of our methods as compared to state-of-the-art approaches, especially when the campaigners are constrained by a budget of a small number of seed nodes, as well as over tree datasets.

In future work, we shall design more efficient algorithms for the host's revenue maximization problem.

Bibliography

[1] Domingos, P., Richardson, M.: Mining the Network Value Customers. In: KDD. (2001)

[2] Kempe, D., Kleinberg, J., Tardos, E.: Maximizing the Spread of Influence through Social Network. In: KDD. (2003)

[3] Lu, W., Bonchi, F., Goyal, A., Lakshmanan, L.V.S.: The Bang for the Buck: Fair Competitive Viral Marketing from the Host Perspective. In: KDD. (2013)

[4] Bharathi, S., Kempe, D., Salek, M.: Competitive Influence Maximization in Social Networks. In: WINE. (2007)

[5] Liu, S., Ying, L., Shakkottai, S.: Influence Maximization in Social Networks: An Ising-model-based Approach. In: Allerton. (2010)

[6] Carnes, T., Nagarajan, C., Wild, S.M., v. Zuylen, A.: Maximizing Influence in a Competitive Social Network: A Follower's Perspective. In: ICEC. (2007)

[7] Lappas, T., Terzi, E., Gunopulos, D., Mannila, H.: Finding Effectors in Social Networks. In: KDD. (2010)

[8] Zehnder, B., Khan, A., Kossmann, D.: Towards Revenue Maximization by Viral Marketing: A Social Network Host's Perspective. In: submitted for publication. (2014)

[9] Leskovec, J., Krause, A., Guestrin, C., Faloutsos, C., VanBriesen, J., Glance, N.: Cost-effective Outbreak Detection in Networks. In: KDD. (2007)

[10] Goyal, A., Lu, W., Lakshmanan, L.V.S.: CELF++: Optimizing the Greedy Algorithm for Influence Maximization in Social Networks. In: WWW. (2011)

[11] Chen, W., Wang, Y., Yang, S.: Efficient Influence Maximization in Social Nerworks. In: KDD. (2009)

[12] Chen, W., Wang, C., Wang, Y.: Scalable Influence Maximization for Prevalent Viral Marketing in Large-Scale Social Networks. In: KDD. (2010)

[13] Wang, Y., Cong, G., Song, G., Xie, K.: Community-based Greedy Algorithm for Mining top-K Influential Nodes in Mobile Social Networks. In: KDD. (2010)

[14] Kim, J., Kim, S.K., Yu, H.: Scalable and Parallelizable Processing of Influence Maximization for Large-Scale Social Networks? In: ICDE. (2013)

[15] Goyal, A., Bonchi, F., Lakshmanan, L.V.S.: A Data-based Approach to Social Influence Maximization. In: VLDB. (2011)

[16] Borgs, C., Brautbar, M., Chayes, J.T., Lucier, B.: Maximizing Social Influence in Nearly Optimal Time. In: SODA. (2014)

[17] Tang, Y., Xiao, X., Shi, Y.: Influence Maximization: Near-Optimal Time Complexity Meets Practical Efficiency. In: SIGMOD. (2014)

[18] Chen, W., Colin, A., Cumming, R., Ke, T., Liu, Z., Rincon, D., Sun, X., Wang, Y., Wei, W., Yuan, Y.: Influence Maximization in Social Networks when Negative Opinions May Emerge and Propagate. In: SDM. (2011)

[19] Budak, C., Agrawal, D., Abbadi, A.E.: Limiting the Spread of Misinformation in Social Networks. In: WWW. (2011)

[20] Li, F.H., Li, C.T., Shan, M.K.: Labeled Influence Maximization in Social Networks for Target Marketing. In: SocialCom/PASSAT. (2011)

[21] Aggarwal, C.C., Khan, A., Yan, X.: On Flow Authority Discovery in Social Networks. In: SDM. (2011)

[22] Cook, S.: The Complexity of Theorem-proving Procedures. In: STOC. (1971)

[23] Kimura, M., Saito, K.: Tractable Models for Information Diffusion in Social Networks. In: PKDD. (2006)

[24] Gabow, H.N., Galil, Z., Spencer, T., Tarjan, R.E.: Efficient Algorithms for Finding Minimum Spanning Trees in Undirected and Directed Graphs. Combinatorica **6**(2) (1986)

[25] Nemhauser, G.L., Wolsey, L.A., Fisher, M.L.: An Analysis of Approximations for Maximizing Submodular Set Functions. Mathematical Programming **14**(1) (1978) 265–294

[26] Potamias, M., Bonchi, F., Gionis, A., Kollios, G.: k-Nearest Neighbors in Uncertain Graphs. PVLDB (2010)

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Eigenständigkeitserklärung

Die unterzeichnete Eigenständigkeitserklärung ist Bestandteil jeder während des Studiums verfassten Semester-, Bachelor- und Master-Arbeit oder anderen Abschlussarbeit (auch der jeweils elektronischen Version).

Die Dozentinnen und Dozenten können auch für andere bei ihnen verfasste schriftliche Arbeiten eine Eigenständigkeitserklärung verlangen.

Ich bestätige, die vorliegende Arbeit selbständig und in eigenen Worten verfasst zu haben. Davon ausgenommen sind sprachliche und inhaltliche Korrekturvorschläge durch die Betreuer und Betreuerinnen der Arbeit.

Titel der Arbeit (in Druckschrift):

Towards Revenue Maximization by Viral Marketing:
A Social Network Host's Perspective

Verfasst von (in Druckschrift):

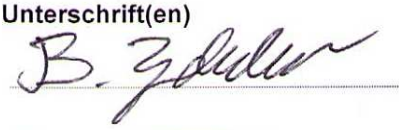
Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich.

Name(n):	**Vorname(n):**
Zehnder	Benjamin Robert

Ich bestätige mit meiner Unterschrift:

- Ich habe keine im Merkblatt „Zitier-Knigge" beschriebene Form des Plagiats begangen.
- Ich habe alle Methoden, Daten und Arbeitsabläufe wahrheitsgetreu dokumentiert.
- Ich habe keine Daten manipuliert.
- Ich habe alle Personen erwähnt, welche die Arbeit wesentlich unterstützt haben.

Ich nehme zur Kenntnis, dass die Arbeit mit elektronischen Hilfsmitteln auf Plagiate überprüft werden kann.

Ort, Datum	**Unterschrift(en)**
Hagendorn, 14.09.2014	

Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich. Durch die Unterschriften bürgen sie gemeinsam für den gesamten Inhalt dieser schriftlichen Arbeit.