Selecting Influential Features by a Learnable Content-Aware Linear Threshold Model

Ansh Khurana
IIT Bombay

Alvis Logins
Aarhus University

Panagiotis Karras
Aarhus University

ABSTRACT

Consider a network in which items propagate in a manner determined by their inherent characteristics or features. How should we select such inherent content features of a message emanating from a given set of nodes, so as to engender high influence spread over the network? This *influential feature set selection* problem has received scarce attention, contrary to its dual, *influential node set selection* counterpart, which calls to select the initial adopter nodes from which a fixed message emanates, so as to reach high influence. However, the influential feature set selection problem arises in many practical settings, where initial adopters are given, while propagation depends on the perception of certain *malleable* message features. We study this problem for a diffusion governed by a *content-aware linear threshold* (CALT) model, by which, once the aggregate weight of influence on a node exceeds a randomly chosen threshold, the item goes through. We show that the influence spread function is not submodular, hence a greedy algorithm with approximation guarantees is inadmissible. We propose a method that learns the parameters of the CALT model and adapt the SimPath diffusion estimation method to build a heuristic for the influential feature selection problem. Our experimental study demonstrates the efficacy and efficiency of our technique over synthetic and real data.

ACM Reference format:
Ansh Khurana, Alvis Logins, and Panagiotis Karras. 2020. Selecting Influential Features by a Learnable Content-Aware Linear Threshold Model. In *Proceedings of Proceedings of the 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19–23, 2020 (CIKM '20),* 10 pages.
https://doi.org/10.1145/3340531.3411886

1 INTRODUCTION

In viral marketing campaigns, organizations aim to spread awareness of new products, ideas, and services across a network. The problem of *Influence Maximization by Node Selection* (IMNS) [17, 24] is to select *seed nodes* from which to initiate a promotion campaign so as to attain the best outcome in terms of nodes reached. Yet the outcome of a network-based promotion campaign depends not only on its starting nodes, but also on the appeal of the promoted innovation, product, idea, or, in general, *meme*. In its turn, the appeal of a meme depends not only on its core features, but also on circumstantial and malleable features of the campaign; one may adapt such features in order to achieve desirable outcomes [8, 9].

The theory of Uses & Gratifications [16] provides insights on how brand posts engage social network users; parameters such as *content type* (e.g., entertaining), *media type* (e.g., vivid), *posting time* (e.g., peak hours) and *valence of comments* (e.g., positive) have an effect on perception [8, 9]. Some works investigate ways to select features that invoke widespread adoption over a network through peer-to-peer effects, based on randomized trials [1], as a complement to the IMNS problem [2], and as a standalone objective [14], namely the problem of *Influence Maximization by Feature Selection* (IMFS). Yet this IMFS problem has been studied only under a *content-aware* variant of the *Independent Cascade* (IC) model [14]; by this model, each neighbor of a node u gets an independent chance to influence u. Besides, the parameters of the diffusion model that governs the process are assumed to be given as input; to our knowledge, no previous work proposes a way to learn such parameters.

In this paper, we propose a novel content-aware diffusion model extending the *Linear Threshold* (LT) model, by which the neighbors of a node u influence u collectively, and study the IMFS problem under model parameters learned from real-world network log data. In this setting, the set of initial adopters is given, while certain *content features* of the propagated meme, such as topics of interest, public persons, locations, and abstract themes [14], are to be chosen so as to maximize the expected number of network nodes it reaches. This setting corresponds to the problem faced by an organization that needs to choose a set of content features that will maximize the expected appeal of a promotion campaign initiated from a fixed set of subscribers. To learn model parameters, we exploit that fact that online social network users are associated to such content features expressed via posting topics. We denote the set of features a user is associated with, or interested in, as *user features.* Our core premise is that, the more content and user features overlap, the more likely a user is to be interested in the diffused item, while the contagion mechanism is equivalent to a process by which a user is influenced by at most one of its incoming neighbors by a probability depending on the weight of the associated edge [17].

We outline our main contributions as follows:

(1) We devise a *Content-Aware Linear Threshold* (CALT) model that governs a network contagion dependent on content features, and study the properties of its spread function.
(2) We study the IMFS problem under the CALT model, show its hardness, and propose heuristic algorithms therefor based on Monte Carlo simulations (MC) and direct spread estimation.
(3) We propose a way to learn the parameters of the CALT model by a credit allocation technique.
(4) We show that our direct-estimation algorithm outperforms the MC-based one in efficiency and scalability without an efficacy disadvantage, on synthetic and learned data.

CIKM '20, October 19–23, 2020, Virtual Event, Ireland

ACM ISBN 978-1-4503-6859-9/20/10…$15.00
https://doi.org/10.1145/3340531.3411886

2 BACKGROUND AND RELATED WORK

We review related work in influence maximization problem variants, algorithms, and diffusion models.

2.1 Influence by node selection

The expected number of nodes reached, or *expected spread*, over a network represented by a di-graph $G(V,E)$ is a *submodular* function of the set of initially activated *seed nodes* $S \subset V$ under both the Independent Cascade (IC) and Linear Threshold (LT) models [17]. Therefore, the problem of selecting a set S that maximizes expected spread, or *Influence Maximization by Node Selection* (IMNS) [17], admits an $(1 - 1/e - \varepsilon)$-approximation algorithm [28], where e is a base of a natural logarithms and ε a positive real number.

By the IC model, starting with seed set S, each node activated in the previous step tries to activate each out-neighbour with a probability assigned to the corresponding edge. On the other hand, by the LT model, the probability that a contagion reaches a node v from its infected in-neighbors depends on a node-specific threshold $\{\theta_v : v \in V\}$, being an i.i.d. uniform random variable on $[0,1]$. This LT mechanism is equivalent to one where each node designates at most one incoming edge as *active*, by a probability equal to that edge's weight, and the contagion goes through active edges [17]. Therefore, by LT, the contribution of a seed node v to expected spread equals the sum of weights of all simple paths originating from v, while the spread of a seed set S equals the sum of contributions of all individual nodes $v \in S$. The state-of-the-art algorithm for IMNS under the LT model, SimPath [12], is based on this *simple path* property. As enumerating all simple paths from nodes in S is #**P**-hard [35], SimPath estimates each node's contribution to expected spread by exploring paths up to a probability threshold η.

2.2 Influence by feature selection

A vast literature has suggested generalizations of the IMNS problem that preserve the submodularity property [24, 26], which allows for sampling-based solutions [5] that scale to networks of millions of nodes. However, this submodularity does not extend to IM problem variants, such as, for example, the problem of adaptively selecting seed nodes under time constraints [34] or sharing ℓ message parts among k seed nodes [23]; another such variant, which concerns us, is the problem of *Influence Maximization by Feature Selection* (IMFS) under a *content-aware* variant of the IC model; this problem is **NP**-hard to approximate withing a factor of $n^{1-\epsilon}$ for any $\epsilon > 0$ [14]. By the *Content-Aware Independent Cascade* (CAIC) model [14], the probability of activation through an edge (u,v) depends on two sets of features, one (F_i) assigned to the diffusion action and another (F_v) to the target node v, and is calculated as $P(u,v) = b_{uv} + q_{uv} \cdot h_{uv}(F_v, F_i)$, $b_{uv}, q_{uv} \in [0,1]$, with $h_{uv} = \min\left\{\frac{1-b_{uv}}{q_{uv}}, |F_v \cap F_i|\right\}$, where b_{uv} is a *base probability* of influence expressing the core strength of the connection, and q_{uv} a *marginal probability* indicating sensitivity to the overlap between action features F_i and node features F_v, under a sanity bound.

Interestingly, the CAIC model on k topics is reducible to the topic-aware model of [3], whereby the diffusion process is associated with a *topic vector* γ^z, where $z = 1 \ldots k$ denotes a feature, and each edge $\{u,v\}$ is associated with a probability p^z_{uv}, indicating the power of feature z across $\{u,v\}$. Under the IC model, the activation probability across edge $\{u,v\}$ is $P(v|u) = \sum_{z=1}^{k} \gamma^z p^z_{uv}$. Under the LT model, the probability that a contagion reaches a node v from its infected in-neighbors is $P(v) = \mathbb{P}\left[\sum_{z=1}^{k} \sum_{u \in \mathcal{N}(v)} \gamma^z p^z_{uv} \geq \theta_v\right]$, where $\mathcal{N}(v)$ is the set of infected in-neighbors of v and θ_v the model's node-specific threshold. We reduce the CAIC model to the topic-aware model of [3] by setting topic-aware vectors over the same k topics, plus a $(k+1)^{\text{th}}$ topic corresponding to the content-independent base probability b_{uv}, as follows:

$$\gamma^z = \begin{cases} 1, & \text{if } z \in F \text{ or } z = k+1 \\ 0, & \text{otherwise} \end{cases} \qquad p^z_{uv} = \begin{cases} q_{uv}, & \text{if } z \in F_v \\ b_{uv}, & \text{if } z = k+1 \\ 0, & \text{otherwise} \end{cases}$$

Despite this reduction of the one to the other, the CAIC model differs from the topic-aware model of [3] by virtue of its *discrete* nature. In CAIC, diffusion actions and nodes are associated with *sets*, hence the problem of optimally configuring a diffusion action is one of set function optimization, on which continuous optimization techniques, such as stochastic gradient descent, are inapplicable. Besides, while [3] and [14] propose similar diffusion models, the one [3] studies the problem of learning model parameters, but not the problem of optimally configuring a diffusion action under that model; in reverse, the other [14] studies the problem of selecting features to optimally configure a diffusion action under its model, but not the problem of learning that model's parameters.

In a different vein, a problem reminiscent of ours is that of finding the top-k most influential topics relevant to a particular keyword query and user [22] or discovered community [21].

2.3 Learning model parameters

Despite the popularity of the IC and LT models [5, 7, 17, 33], the question of deriving their parameters from real-world data has been scarcely studied [25]. Saito et al. [32] proposed an Expectation-Maximization approach (EM) to train the IC model parameters, while Goyal et al. [10] suggested a more scalable approach, estimating the edge probabilities of the IC model as the fraction of common actions of two adjacent nodes over the number of source node actions, as well as a learning approach for the General Threshold (GT) model, of which IC and LT are special cases, while embedding a temporal decay factor. By the GT model, a node has a general activation function f that aggregates probabilities from all its in-neighbors, and the result is compared to a randomly chosen threshold θ to decide about the node's activation. The suggested training approach is based on the *credit assignment* principle, which assigns uniformly, or, more generally, distributes [11], the credit for each successful node activation among all candidate influencers.

In another direction, Barbieri et al. [3] proposed an Expectation-Maximization (EM) approach for two special cases of GT model, the IC and a newly proposed *Air* model, where the the activation function f is a logistic function applied to a linear combination of incoming probabilities. In these two cases, it is possible to derive a closed form of the Complete-Data Expectation Likelihood function, required for EM to work. However, such a closed form under the LT model remains unknown. We extend this credit assignment principle to a *content-aware* generalization of the LT model; to our knowledge, our work is the first to propose a parameter learning framework for a content-aware diffusion model.

3 PROBLEM STATEMENT

We formulate the problem of influence maximization by feature selection under the LT model, over a network expressed as a directed graph $G = (V, E)$, where $V = \{v_1, v_2, ..., v_n\}$ is a set of nodes, each of which corresponds to an individual user, and $E \subset V \times V$ is a set of directed edges representing social relations among users. First, we outline our diffusion model.

Content-aware LT model. An *action* is a set of user messages exchanged over a network and forming a single instance of diffusion. Let $F = \{f_1, \dots, f_k\}$ be a set of features (i.e., topics or interests) associated with actions and users. We define the *Content-Aware Linear Threshold* (CALT) model as a generalization of the LT model, in which edge weights depend on action and user features as follows:

$$f_{uv} = b_{uv} + q_{uv} \cdot h_{uv}(F_v, F_i),\ b_{uv}, q_{uv} \in [0, 1] \tag{1}$$

$$h_{uv}(F_v, F_i) = \min \left\{ \frac{1 - b_{uv}}{q_{uv}}, |F_v \cap F_i| \right\} \tag{2}$$

where u is in-neighbour of v; b_{uv} is a *base* weight indicating the influence of u on v, and q_{uv} a *quotient* indicating how much that influence may increase due to message features; F_v is a set of topics that node v is interested in, F_i is a set of topics that characterize the ith propagation action. The activation probability of a node v is

$$P(v) = \mathbb{P}\left[\sum_u f_{uv} \geq \theta_v\right]$$

where $\theta_v \in [0, 1]$ is the LT model's random node-specific threshold fixed at the beginning of propagation. The nodes activated during a propagation action are reachable along activated edges from the set S of initially active *seed* nodes. Computing the expected number of nodes reached by, or *expected spread* of, an action with feature set F and seed set S, $\sigma(S, F)$, is #**P**-hard, due to a trivial reduction of CALT to the regular LT model.

Edge weights	f_{uv}
Edge weight parameters	b_{uv}, q_{uv}
in-degree of vertex v	d^v_{in}
Complete feature universe	$\mathcal{F}$
Feature set of the ith action and a vertex v	F_i, F_v
Sanity bound function	$h(\cdot)$
Expected spread	$\sigma(S, F)$
Size of feature set	k
Random node-specific activation threshold	θ_v
Weighted cascade weight scaling factor	α
Binary threshold parameter	Θ
Accuracy parameter for SIMPATH	η

Table 1: Notations

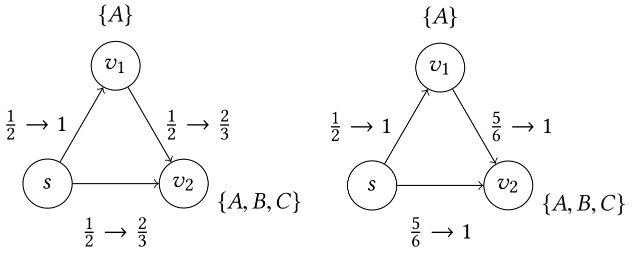
Table 1 gathers our basic notations. Theorem 3.1 shows that $\sigma(S, F)$ does not admit submodularity-based approximation guarantees.

THEOREM 3.1. *The spread function $\sigma(S, F)$ under the CALT model is neither submodular nor supermodular.*

PROOF. Consider a network G with nodes $V = \{s, v_1, v_2\}$, illustrated on Figure 1, where s is a seed node. The expected activation probability of a node v in the CALT model, as in the LT model [12], is equal to the sum of probabilities of all paths from any seed node to v. Thus, the expected spread of seed node s with feature set F is:

$$\sigma(s, F) = f_{sv_1} + f_{sv_2} + f_{sv_1} f_{v_1 v_2}$$

where f_{uv} is the weight of edge (u, v) that depends on parameters b_{uv}, q_{uv}, F, and F_v, according to Equation 1.


$\{A\}$
v_1
$\frac{1}{2} \longrightarrow 1$
$\frac{1}{2} \longrightarrow \frac{2}{3}$
s
v_2
$\{A, B, C\}$
$\frac{1}{2} \longrightarrow \frac{2}{3}$
$\{A\}$
v_1
$\frac{1}{2} \longrightarrow 1$
$\frac{5}{6} \longrightarrow 1$
s
v_2
$\{A, B, C\}$
$\frac{5}{6} \longrightarrow 1$

(a) $F_1 \longrightarrow F_1 \cup \{A\}$, $F_1 = \emptyset$ **(b)** $F_2 \longrightarrow F_2 \cup \{A\}$, $F_2 = \{B, C\}$

Figure 1: Counterexample for submodularity of σ; edges are labeled by weight updates after adding a new feature A.

Let Δ be the marginal gain brought by adding a new feature f:

$$\Delta(F_1, f) = \sigma(S, F_1 \cup \{f\}) - \sigma(S, F_1)$$

$\sigma(S, F)$ is submodular (supermodular) with respect to F if and only if it satisfies the property of nonincreasing (nondecreasing) returns:

$$\forall F_1, F_2, f\ :\ F_1 \subseteq F_2, f \notin F_2 \rightarrow \Delta(F_1, f) \geq (\leq)\, \Delta(F_2, f)$$

The feature space is $\mathcal{F} = \{A, B, C\}$. We consider two cases of marginal case comparison. Figure 1 presents the first case, where $F_{v_1} = \{A\}$ and $F_{v_2} = \{A, B, C\}$, while all edges have $b_{uv} = \frac{1}{2}$, and $q_{sv_1} = \frac{1}{2}$ and $q_{sv_2} = q_{v_1 v_2} = \frac{1}{6}$. Then:

$$f_{sv_1} = 1/2 + 1/2 \cdot |\{A\} \cap F_i| \tag{3}$$

$$f_{sv_2} = 1/2 + 1/6 \cdot |\{A, B, C\} \cap F_i| \tag{4}$$

$$f_{v_1 v_2} = 1/2 + 1/6 \cdot |\{A, B, C\} \cap F_i| \tag{5}$$

We assume sets $F_1 = \emptyset$, $F_2 = \{B, C\}$, where $F_1 \subset F_2$, and added feature $f = A$; then the marginal gain for F_1 is:

$$\Delta(F_1, A) = \sigma(s, A) - \sigma(s, \emptyset) = \\ (1 + 2/3 + 1 \cdot 2/3) - (1/2 + 1/2 + 1/2 \cdot 1/2) = 13/12 \tag{6}$$

while the marginal gain for F_2 is:

$$\Delta(F_2, A) = \sigma(s, \{A, B, C\}) - \sigma(s, \{B, C\}) = \\ (1 + 1 + 1 \cdot 1) - (1/2 + 5/6 + 1/2 \cdot 5/6) = 15/12 \tag{7}$$

Figures 1a and 1b illustrate the updates in weights for F_1 and F_2, respectively. We observe that $\Delta(F_2, A) > \Delta(F_1, A)$, hence the spread function is *not* submodular.

In the second case, we consider $F_{v_1} = F_{v_2} = \{A, B\}$ and all edges having $b_{uv} = \frac{1}{2}$, and $q_{uv} = \frac{1}{2}$, hence:

$$f_{uv} = 1/2 + 1/2 \cdot \min\{1, |\{A, B\} \cap F_i|\}$$

Let $F_1 = \emptyset$, $F_2 = \{B\}$ and $f = A$. Now the marginal gains are

$$\Delta(F_1, A) = \sigma(s, \{A\}) - \sigma(s, \emptyset) = 3 - 5/4 = 7/4$$

and

$$\Delta(F_2, A) = \sigma(s, \{A, B\}) - \sigma(s, \{B\}) = 3 - 3 = 0$$

Thus, $\Delta(F_2, f) < \Delta(F_1, f)$, hence σ is *not* supermodular either. We conclude that σ is neither submodular nor supermodular. □

The **problem** of *Influence Maximization by Feature Selection* (IMFS) is to select a set $F \subset \mathcal{F}$ of up to k features that maximizes the expected spread for a given seeds S under the CALT model:

$$\max_{F:|F| \leq k} \sigma(S, F) \tag{8}$$

4 SOLUTIONS

The probability that a node's weight $f_v \in [0, 1]$ exceeds an i.i.d. uniform random threshold θ_v in $[0, 1]$ is equal to f_v. Therefore, on a graph where each node has only one incoming edge, the IMFS problem under that CALT model is reduced to the IMFS problem under the IC model, which is **NP**-hard to approximate within a factor of $n^{1-\varepsilon}$ [14]. Given this hardness, we design heuristic solutions for the IMFS problem. We first present a greedy baseline that estimates influence spread by Monte-Carlo simulations; then, we propose a more efficient heuristic that employs a spread estimation technique adopted from the SimPath algorithm [12].

4.1 Greedy baseline

Algorithm 1 presents our Greedy baseline, which iteratively selects the feature in $\mathcal{F}$ incurring the highest marginal gain in expected spread in a network G with seed set S, as calculated by Monte-Carlo simulations; this baseline follows previous work in the area [14].

Algorithm 1: Greedy

```
Input: $G(V, E)$, $S$, $\mathcal{F}$
$F = \emptyset$;
while $|F| < k$ do
    for every $f \in \mathcal{F} \setminus F$ do
        calculate $\sigma(F + \{f\})$ using Monte Carlo simulations;
    $F = F \cup \arg\max_f \{\sigma(F + \{f\})\}$;
return $F$;
```

4.2 SimPath-based heuristic

We propose a feature selection algorithm based on the SimPath spread estimation technique [12]. Our algorithm maintains the structure of Algorithm 1: we greedily pick k features, while evaluating expected spread for each feature. However, as the evaluation of expected spread by Monte-Carlo simulations is computationally demanding, we opt for estimating spread using the SimPath algorithm [12], which we explain in the following.

The SimPath algorithm is based on a series of observations regarding the LT model. First, the spread of a given seed set S is equal to the contribution of each seed node s to the spread on the induced subgraph of G with vertex set $(V \setminus S) \cup \{s\}$. This property transfers to the CALT model, as the following theorem shows.

Theorem 4.1. *The expected spread of seed set S by the CALT model is the sum of expected spreads over all nodes $s \in S$, each on the induced subgraph with vertex set $(V \setminus S) \cup \{s\}$. That is,*

$$\sigma(S) = \sum_{s \in S} \sigma^{(V \setminus S) \cup \{s\}}(s) \tag{9}$$

Proof. The statement of the theorem holds for the LT model due to result of Goyal et al. [12]. The CALT model is equivalent to the LT model for each particular action feature set F_i. Since the feature set F_i does not change during the diffusion process, the theorem still holds. □

Secondly, as noted in Theorem 3.1, the expected spread from a single seed s is equal to the sum of probabilities of all paths from s to other nodes [12]. Lastly, in practice, edge probabilities, f_{uv}, are generally small, hence the probability of a path tends to rapidly decrease with a path length. Based on these three observations, we estimate expected spread under the CALT model using simple paths with a probability threshold η, as the SimPath algorithm does [12].

Algorithm 2 shows the SimPath spread estimation procedure. For a seed set S and path probability *pruning threshold* η, we evaluate the spread independently for each $s \in S$ on a graph, induced by s, with nodes $(V \setminus S) \cup \{s\}$, and sum the results to get the final estimate.

Algorithm 2: CA-SimPath spread estimation routine

```
Input: $G(V, E)$, $S$, $\mathcal{F}$, $\eta$
$\sigma = 0$;
for $s \in S$ do
    $\sigma = \sigma + $ ForwardBacktrack$(G, s, (V \setminus S) \cup \{s\}, \eta)$;
return $\sigma$
```

Algorithm 3 shows how we estimate the spread of a single seed node s. We enumerate all simple paths in a neighborhood of s, in which the probability of any path, calculated as the product of edge weights along the path, is not lower that the threshold η.

In Line 1 we create a few variables: Q is a stack that contains the nodes of the current path, initiated with the seed node s; D is a map that stores a set of out-neighbors for each node that has been considered so far; p stores the current path's probability and σ stores the current spread estimate.

The algorithm builds path segments from each node in Q by a look-forward procedure, ensuring that it does not create cycles and new segments have not been previously explored; Lines 4–8 perform this procedure in a loop. Upon the loop's termination, i.e., when it reaches a path's end or go under the threshold η, it *backtracks* by removing the last node from the stack (Line 9), and recovering the current path's weight to its previous state (Line 10). Once Q is empty, the algorithm returns its spread estimate σ. Higher values of η imply *lower* runtime and *less* accurate local spread estimation; $\eta = 0$ results to an exact local spread computation.

Algorithm 3: ForwardBacktrack

```
   Input: $G, s, W, \eta$
1  $Q \leftarrow \{s\}$; $D \leftarrow$ Null; $p \leftarrow 1$; $\sigma \leftarrow 1$ ;
2  while $Q \neq \emptyset$ do
3      $u \leftarrow Q$.last();
4      while $\exists v \mid (u, v) \in E, v \notin Q, v \notin D[u], v \in W$ do
5          $D[u].insert(v)$;
6          if $p \cdot f_{uv} \geq \eta$ then
7              $Q.add(v)$;
8              $p \leftarrow p \cdot f_{uv}$; $\sigma \leftarrow \sigma + f_{uv}$; $u \leftarrow v$;
9      $v \leftarrow Q.last()$; $Q \leftarrow Q - v$;
10     $p \leftarrow p / f_{uv}$;
11 return $\sigma$;
```

5 CALT TRAINING

In this section we present or approach for training the CALT model using real-world data.

5.1 Credit assignment in the LT model

As discussed in Section 2.3, Goyal et al. [10] presented the *credit assignment* algorithm for training the General Threshold model, of which LT is a special case. The algorithm is based on the assumption that all relevant in-neighbours of a node share the same *credit* for each *successful* activation of that node. For example, if two nodes have posted a message about Trump, and then a common neighbour of theirs also posts about Trump, then each of the former two nodes is considered 50% responsible for activating the third node. Note that, by this principle, nodes share credits for successful actions in a manner agnostic to whether the ground-truth edge weights are the same or not.

Formally, let $t(a)$ be a timestamp of an action a. We consider an action a performed by node u as *adapted* by a neighbor node v if v performs the actions *after* u, with timestamps $t_u(a) < t_v(a)$. For the regular LT model, the credit assignment algorithm determines edge weigths (or influence probabilities) as follows:

$$f_{uv} = \frac{\sum_a credit_{uv}(a)}{|A_u|}$$

where a is an action propagated in one diffusion instance (*cascade*) and A_u the set of all actions of u. The credit is defined as:

$$credit_{uv}(a) = \frac{1}{\sum_{w \in N} I\left(t_w(a) < t_v(a)\right)}$$

where I is an indicator function that an in-neighbour w of a node v has been activated by an action a before v itself.

5.2 Credit assignment in the CALT model

The parameters of the CALT model are b_{uv}, q_{uv}, F_v and F_i. Out of those, the last two (i.e., user preferences and topics of actions) are discrete sets that need to be given as input, or be determined in a preprocessing step (Section 6) from log data, while F_i is the object of optimization in the IMFS problem. Here, we propose a novel *Content Aware Credit Assignment* (CACA) algorithm that efficiently learns the parameters b_{uv} and q_{uv} from log action data.

The CACA algorithm uses the *equal credit share* assumption, by which, for each successful action, an in-neighbour takes $\frac{1}{d}$ of the credit, where d is the total number of in-neighbours that share credit for that action. However, instead of a single parameter value f_{uv}, we need to estimate coefficients b_{uv} and q_{uv}, which are dependent on the features (or topics) of each observed action. For that purpose, we use the *Ordinary Least Squares* (OLS) estimator. Let N be a set of in-neighbours of v, A_u the set of all actions performed by u, and A_{uv} the set of actions performed by u and adapted by v. Then, for each $a \in A_u$, the equal share assumption implies that:

$$b_{uv} + q_{uv}\alpha_v(a) = \mathbb{E}_a\left[\frac{I(t_u(a) < t_v(a))}{\sum_{w \in N} I(t_w(a) < t_v(a))}\right] \tag{10}$$

where $I(t_u(a) < t_v(a))$ indicates that v adopted action a at any time after u, $\sum_{w \in N} I(t_w(a) < t_v(a))$ is the number of in-neighbours who could have influenced v if v has adopted a, and $\alpha_v(a) = |F_v \cap F_a|$ shows how many topics (features) in a match the preferences of v.

Let $d_v(a) = \sum_{w \in N} I(t_w(a) < t_v(a))$, and $n_u = |A_u|$. Then, the OLS solution is:

$$q_{uv} = \frac{\sum_{a \in A_{uv}} \alpha_v(a)\frac{1}{d_v(a)} - \frac{1}{n_u}\left(\sum_{a \in A_u} \alpha_v(a))(\sum_{a \in A_{uv}} \frac{1}{d_v(a)}\right)}{\sum_{a \in A_u} \alpha_v^2(a) - \frac{1}{n_u}\left(\sum_{a \in A_u} \alpha_v(a)\right)^2} \tag{11}$$

$$b_{uv} = \frac{1}{n_u}\sum_{a \in A_{uv}} \frac{1}{d_v(a)} - q_{uv}\frac{1}{n_u}\sum_{a \in A_u} \alpha_v(a) \tag{12}$$

These equations derive from the OLS method applied on a linear equation of the form

$$y_i = \alpha + \beta x_i + \epsilon_i$$

where ϵ_i is random error and

$$y_i = \frac{I\left(t_u(a) < t_v(a)\right)}{\sum_{w \in S} I\left(t_w(a) < t_v(a)\right)}$$
$$x_i = \alpha_v(a),\ \alpha = b_{uv},\ \beta = q_{uv}.$$

5.3 The CACA algorithm

In order to compute coefficients of CALT according to Equations 12 and 11, we have to know the following statistics:

$$n_u = |A_u|$$
$$d_v(a) = \sum_{w \in N} I\left(t_w(a) < t_v(a)\right)$$
$$C^1_{uv} = \sum_{a \in A_{uv}} \alpha_v(a)\frac{1}{d_v(a)},\ C^2_{uv} = \sum_{a \in A_u} \alpha_v(a)$$
$$C^3_{uv} = \sum_{a \in A_{uv}} \frac{1}{d_v(a)},\ C^4_{uv} = \sum_{a \in A_u} \alpha^2_v(a)$$

Then, coefficients are equal to

$$b_{uv} = \frac{1}{n_u}C^3_{uv} - q_{uv}\frac{1}{n_u}C^2_{uv} \tag{13}$$

$$q_{uv} = \frac{C^1_{uv} - \frac{1}{n_u}C^2_{uv}C^3_{uv}}{C^4_{uv} - \frac{1}{n_u}(C^2_{uv})^2} \tag{14}$$

Algorithm 4 presents the CACA algorithm, which utilizes a log of actions on a graph sorted in in chronological order. For each observed action a, the algorithm traverses logs in chronological order considering each *message m* that corresponds to an action a by node v, updating a *currentTable* data structure. If the user-action pair $\{v, a\}$ already exists in *currentTable*, with any time step t, then we just update the time step in the table to that of the currently considered message, t_v (Line 7). The underlying rationale is that any node that the algorithm will consider in future iterations is a potential *target* node, while the current node v is a potential *source node*; it is beneficial to update v's time step to the latest possible.

In case the pair $\{v, a\}$ is not in the table, then we consider message m as the first instance when node v performs action a and proceed to update the values of C^1_{uv}, C^2_{vw}, C^3_{uv}, and C^4_{vw}. We update the n_v value accordingly (Line 9), and, for each *outgoing edge* (v, w), we compute $\alpha_w(a)$ and store it in cache (Line 11), so that we can later use it for updating C^1_{uv}, and duly update C^2_{vw} and C^4_{vw}; as these two are defined as a sum over *all* actions ever performed by a source node v, we may update them as soon as we detect any node performing any action for the first time. On the other hand, to update C^1_{uv} and C^3_{uv}, we need to find all source nodes in *currentTable* that may have successfully influenced v within a time threshold τ. For each such u, we increment $d_v(a)$ and save u in a temporary list *parents* (Lines 15–18). After collecting all such parent nodes, we traverse the list, and update C^1_{uv} and C^3_{uv} for each u that has v as a successful follower. Lastly, we add $< v, a, t_v >$ in the table. After running Algorithm 4, we assign coefficients to each edge according to Equations 13 and 14.

Algorithm 4: Content Aware Credit Assignment (CACA)

```
    Input: A set of messages grouped as actions, sorted by $t$
    Output: $C^{1,2,3,4}_{uv}$, $n_u$ for all $u$ and $v$
 1  for $u, v \in V$ do
 2      $n_v \longleftarrow 0$, $C^{1,2,3,4}_{uv} \longleftarrow 0$;
 3  for $a \in actions$ do
 4      $currentTable \longleftarrow \emptyset$;
 5      for each $< v, m, t_v > | m \in a$ in chronological order do
 6          if $< v, a, t > \in currentTable$ then
 7              $t \longleftarrow t_v$;
 8          else
 9              Increment $n_v$;
10              for $w \in V | (v, w) \in E$ do
11                  Compute $\alpha_w(a)$ and store in cache;
12                  $C^2_{vw} += \alpha_w(a)$;
13                  $C^4_{vw} += \alpha^2_w(a)$;
14              $parents \longleftarrow \emptyset$, $d_v(a) \longleftarrow 0$;
15              for $< u, a, t_u > \in currentTable \wedge (u, v) \in E$ do
16                  if $\tau > t_v - t_u > 0$ then
17                      Increment $d_v(a)$;
18                      Insert $u$ in $parents$;
19              for $u \in parents$ do
20                  $C^1_{uv} += \alpha_v(a) \frac{1}{d_v(a)}$;
21                  $C^3_{uv} += \frac{1}{d_v(a)}$;
22              Add $< v, a, t_v >$ to $currentTable$;
```

5.4 Evaluating efficacy

We evaluate the efficacy of the CACA algorithm by the Area Under ROC Curve (AUC) measure. We define a *positive propagation instance* as a case where a message m exists for node v and a message m' similar to m exists for *any* of the in-neighbours of v. To calculate such positive instance, we traverse all graph nodes, and collect the in-neighbor messages of each node. For each thus collected message, in arbitrary order, we compose the set of in-neighbours that have posted similar messages, and, using that set, we calculate the predicted probability of the influence; in case that value exceeds a predefined threshold μ, we consider the prediction to be positive, otherwise negative. Thereafter, we discard all considered messages. This way, we calculate the True Positive and False Positive Rates for each value of μ, as TPR $= \frac{TP}{TP+FN}$, FPR $= \frac{FP}{FP+TN}$. The AUC is AUC $= \int$ TPR dFPR.

6 DATA PREPARATION

We experiment on diverse network structures and ways of deriving their CALT model parameters, which we discuss in this section.

6.1 Barabási-Albert synthetic networks

We use Barabási-Albert (BA) networks as synthetic network data. Such networks have high clustering coefficients and power-law degree distribution, hence are reasonable imitations of real-world social networks. In particular, we use the algorithm of Holme and Kim [13], available in the NetworkX Python library[1], which extends the original Barabási-Albert model with a *triad formation step*, yet denote it by the BA label, which represents its basis; this algorithm first randomly creates m edges for each node in a graph by *preferential attachment*, as the BA model, and, for each created edge (v, w), adds, with a probability p, an extra edge from v to one of w's neighbors, making a triangle. We use parameter values $p = 0.2$ and $m = 15$. We generate edge weight parameters according to the *trivalency* model. For each edge, we pick uniformly at random one of three values per parameter:

$$b_{uv} \in \{0.1, 0.01, 0.001\},\ q_{uv} \in \{0.1, 0.01, 0.001\}$$

We set the feature space size $|\mathcal{F}|$ and randomly draw features with probability $p = 0.2$ to create node feature sets F_v.

6.2 Gnutella network

Gnutella is a snapshot of the Gnutella peer-to-peer file sharing network [30] with 10,876 nodes and 39,994 edges. To derive CALT model coefficients therefor, we build upon the *Weighted Cascade model* [6, 14], which defines edge weights, in the context of the IC model, as $\frac{1}{d^v_{\text{in}}}$, where d^v_{in} is the in-degree of node v. We adapt a *scaled* weighted cascade model to the CALT model by introducing a hyperparameter c, which indicates how much the spread grows from when features do not match to when they match completely. The boundary conditions for f_{uv} are as follows:

$$f_{uv} = \begin{cases} b_{uv}, \text{if } F_i = \emptyset \\ c \cdot b_{uv}, \text{if } F_i = F_v \end{cases} \tag{15}$$

where b_{uv} follows the scaled Weigted Cascade model, designed so as to obtain realistic overall spread values:

$$b_{uv} = \min \left\{ \alpha b_{uv}, \frac{1}{d^v_{\text{in}}} \right\} \tag{16}$$

where α is a tunable parameter. In our experiments, we set $c = 5$ and $\alpha = 0.2$. We further normalize edge weights so that $\sum_u f_{uv} \leq 1$ on each v. To achieve that, we apply a weighted Sigmoid function $g(\cdot)$ on the edge weight function f_{uv} (Equation 1) and derive a new expression for *normalized* edge weights f'_{uv}:

$$f'_{uv} = \frac{1}{d^v_{\text{in}}} \cdot g\left(b'_{uv} + q'_{uv}|F_v \cap F|\right) \tag{17}$$

where $g(\cdot)$ is the Sigmoid function, while b'_{uv} and q'_{uv} are unknowns; it follows that $\sum_u f'_{uv} \leq \sum_{u \in \text{in-neighbors}(v)} \frac{1}{d^v_{\text{in}}} \leq 1$. Applying the boundary condition in Equation 15 to the normalized

[1] https://networkx.github.io/

edge weights f'_{uv}, we obtain the following closed-form expressions for the parameters b'_{uv} and q'_{uv}:

$$b'_{uv} = \log\left(\frac{b_{uv} d^{v}_{\text{in}}}{1 - b_{uv} d^{v}_{\text{in}}}\right) \tag{18}$$

$$q'_{uv} = \frac{1}{|F_v|} \log\left(\frac{c - c b_{uv} d^{v}_{\text{in}}}{1 - c b_{uv} d^{v}_{\text{in}}}\right) \tag{19}$$

Lastly, we define node feature sets as in the BA case.

6.3 VK network

We use the *VK* dataset presented by Logins et al. [25], a weakly connected subgraph of the VKontakte social network[2] with 2452 nodes and 28108 edges. This dataset comes along with a list of up to 100 latest text posts for each user, 106,217 posts in total. We derive *feature vectors* for these messages as follows. We apply stemming, lemmatization, and remove stopwords, and then vectorize the messages by the *enriched TFIDF* technique [18], which helps to represent very short messages; it first collects the regular *term frequency-inverse document frequency* (TFIDF) statistic, and then updates zero-valued entries in message vectors to new values that depend on the similarity of the corresponding terms to existing terms in the document, which we obtain using the skip-gram model pretrained on Wikipedia. Yet the feature space resulting from this NLP analysis of user messages is quite large ($k = 6.6 \cdot 10^4$); therefore, we first sparsify the resulting TFIDF matrix by taking the top-100 elements per row and setting others to zero, and then apply the Latent Dirichlet Allocation (LDA) dimensionality reduction technique [4] to reduce the feature space size $|\mathcal{F}|$ to 100.

Still, the IMFS problem requires the feature vectors characterizing users and actions (i.e., messages) to be *discrete*. Therefore, we associate each post (i.e., text message) with a *binary* feature vector F_i using a threshold parameter Θ:

$$F_i = \{f_1, .., f_{|\mathcal{F}|}\} = \{I(t_j > \Theta)\}$$

where $I(\cdot)$ is an indicator function and t_j is the j-th topic of a message. We tune Θ experimentally to $\Theta = 0.034$, so that adding features creates contagion on the VK network. Similarly, we define the topic vector of each user u as the thresholded average over all messages u has authored:

$$F_u = \left\{ I\left(\frac{1}{|M|} \sum_{M} t_j > \Theta \right) \right\}$$

where M is a set of topic vectors of messages authored by u.

To apply the CALT training method of Section 5, we need to define when a post of a node v constitutes an instance of propagation (i.e., repost) of a post of a neighbor node u; reposts are not tagged. We thus define three criteria to identify reposts, as follows:

- the two posts are *subsequent* in time;
- their *time difference* is smaller than a threshold τ;
- the posts have *similar* content.

In experiments, we set τ to one month, which is reasonable, since user influence in a social network may persist up to 100 weeks [10]. We assume each node is *influenced* on a topic only once, but may post about the same topic several times. Thus, when we assess a node v as a *target* node, we consider the time when v performs an action a as the time of the *earliest* message of v associated with F_a; when we assess a node u as a *source* node, we consider the time step of *any* message of u associated with F_a. Thus, an action a may be associated with a *set* of time steps. Lastly, to determine which messages have *similar* content, we employ a *natural k-means clustering* of VK messages; we opt for $k = 82$ clusters, with which we obtain the maximum Silhouette Score [31], as we observe in Figure 2, and consider messages that belong to the same cluster as being of the same action.

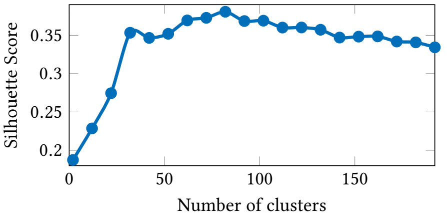
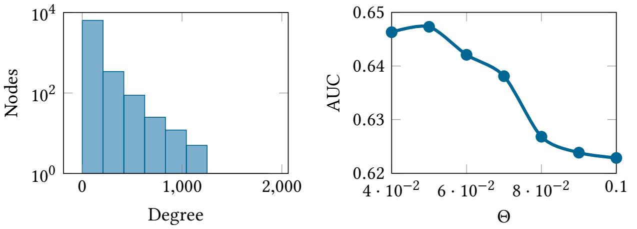


Figure 2: Clustering quality of VK messages



(a) Degree distribution **(b) Selection of the Θ parameter.**

Figure 3: Citation dataset statistics.

6.4 Citation network

As a third real-world dataset, we use the Arxiv high energy physics theory citation graph [20], which contains 27,770 papers (nodes) with 352,807 citations (directed edges), and paper metadata. First, we select a weakly connected component of papers that contain title, authors, and abstract in their metadata, ending up with a network of 16,890 nodes and 166,812 edges. From these data we induce a network in which nodes represent authors, and edges represent pairs of coauthors of a common paper, as well as citations by one author to another, obtaining a network of 6,852 nodes and 208,467 edges with power-law degree distribution as Figure 3a shows.

To learn the parameters of influence within the high-energy physics community represented by this network by the method of Section 5, we assume that each paper is an action (i.e., message), and each *citation* is an instance of successful propagation from all authors of the source paper to all authors of the target paper, i.e., v performs the *same action* as u if a paper of v cites a paper of u. We define binary feature vectors for papers (i.e., messages) and authors (i.e., nodes) similarly to the way we do so on the VK data: we apply stemming, lemmatization, and *regular* TFIDF on paper metadata (i.e., merged title and abstract), and reduce the dimensionality of the resulting TFIDF matrix to $|\mathcal{F}| = 100$ features by LDA. We render the resulting feature vectors binary using the threshold value $\Theta = 5 \cdot 10^{-5}$ that maximizes the AUC measure (see Section 5.4) under cross-validation with a 1:3 split, as Figure 3b shows. As in VK, the papers of an author u define its topic vector.

[2] https://vk.com/

7 EXPERIMENTAL STUDY

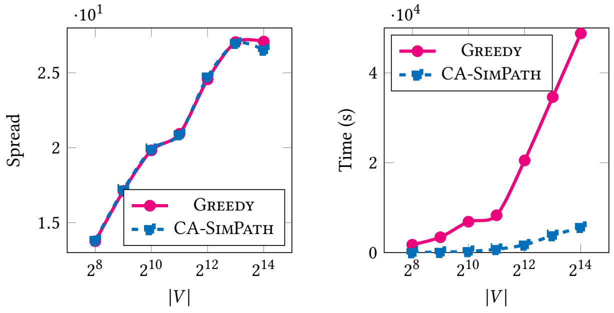
Here, we present our experimental evaluation of IMFS algorithms Greedy and CA-SimPath on the data presented in Section 6.

7.1 Setting

Our feature selection algorithms are implemented[3] in C++, compiled using gcc 7.4.0 with the -O3 flag. We implemented the algorithms for model training in Python, and ran all experiments on an Intel Xeon CPU E5-2687W v3 @ 3.10GHz machine with 377GB RAM running Ubuntu 18.04.

Seed selection As the seed set S is an input parameter, we select seeds uniformly at random from the same community on all real-world data, as in [14]. We define a *community* of users U as those who share an interest in a particular topic f, $U = \{u | f \in F_u\}$, picking f at uniformly random. In all experiments, we select 20 seed nodes, unless otherwise specified. On synthetic data, we select a set of five seed nodes with the largest out-degrees, following the naïve degree heuristic for influence maximization, so as to evaluate our algorithms in an informative setting.

Monte-Carlo evaluation We estimate the final expected spread we obtain with any solution via 10,000 Monte-Carlo iterations; we exclude the time for this estimation from runtime results.



(a) Spread

(b) Runtime

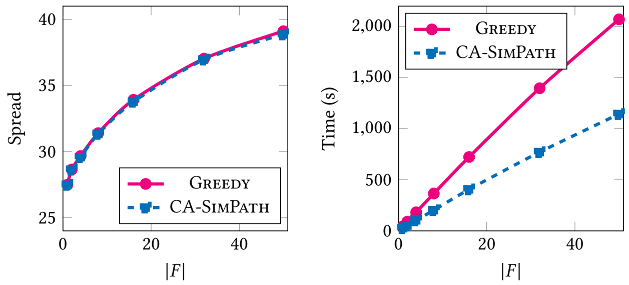
Figure 4: Performance on BA networks, $\eta = 10^{-5}$.

7.2 Results on BA synthetic data

We first study the obtained expected spread value and runtime on the synthetic BA network with growing network size, using the top-5 nodes of largest degree as seed set and path probability threshold $\eta = 10^{-5}$. As Figure 4 shows, the spread increases with network size $|V|$. Critically, the runtime of CA-SimPath scales much more gracefully than that of Greedy with network size, without deterioration in solution quality. For the network size $|V| = 2^{14}$, Greedy slightly outperforms CA-SimPath.

7.3 Results on Gnutella

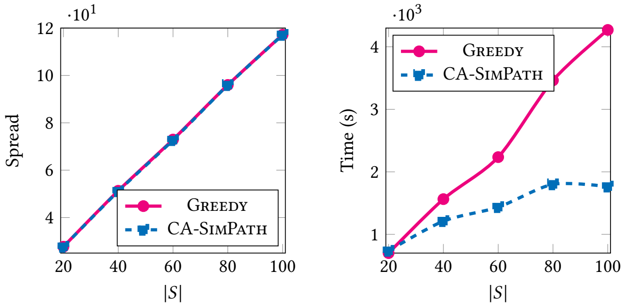
Figures 5 and 6 present our results on the Gnutella dataset, measuring spread and runtime against feature set size $|F|$ and seed set size $|S|$. We set scaling parameter $\alpha = 5$, weight parameter $c = 0.2$, and probability threshold $\eta = 10^{-5}$, and define seed sets by picking random nodes from a group sharing a randomly chosen feature. Remarkably, once again, the spread values for both algorithm overlap, while the runtime of CA-SimPath scales much more gracefully than that of Greedy; as the seed set size grows from 20 to 100, the runtime of Greedy rises by 509%, while that CA-SimPath rises by 140% and yields practically the same spread.



(a) Spread

(b) Runtime

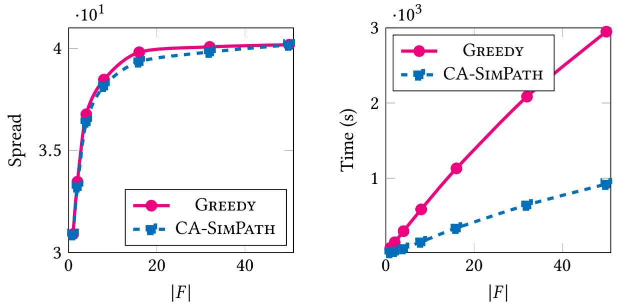
Figure 5: Performance on Gnutella vs. $|F|$, $\eta = 10^{-5}$.



(a) Spread

(b) Runtime

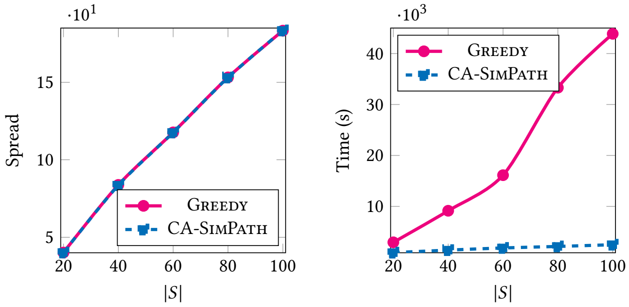
Figure 6: Performance on Gnutella vs. seed set size, $\eta = 10^{-5}$.



(a) Spread

(b) Runtime

Figure 7: Performance on VK vs. $|F|$, $\eta = 10^{-4}$.



(a) Spread

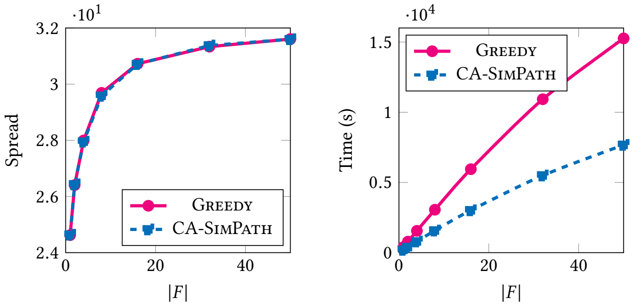
(b) Runtime

Figure 8: Performance on VK vs. seed set size, $\eta = 10^{-4}$.

[3] The code is available at https://github.com/AnshKhurana/CAIM.

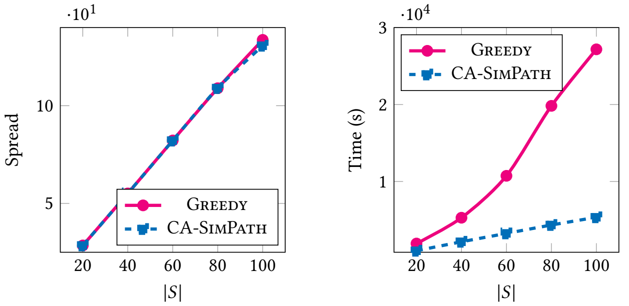
7.4 Results on VK

We now proceed our study to the VK dataset; we set $\eta = 10^{-4}$ and, as discussed in Section 6.3, $\Theta = 0.034$, which renders the network responsive to feature addition. For instance, after adding 50 features on a network of 100 seed nodes, we observe a marginal influence spread gain of 0.0079 and 0.0107 when adding one more feature with the Citation and Gnutella data, respectively, and a much higher value of 0.0747 with the VK data. The results in Figures 7 and 8 show that CA-SimPath is significantly more efficient than Greedy. In a middle range of feature set size values Greedy achieves slightly better spread value. With larger seed set sizes, though, the execution time of Greedy increases more than 13-fold, while that of CA-SimPath by only 2.5. This results showcases that CA-SimPath achieves scalability without compromising its efficacy.



(a) Spread **(b) Runtime**

Figure 9: Performance on Citation vs. $|F|$, $\eta = 0.05$.



(a) Spread **(b) Runtime**

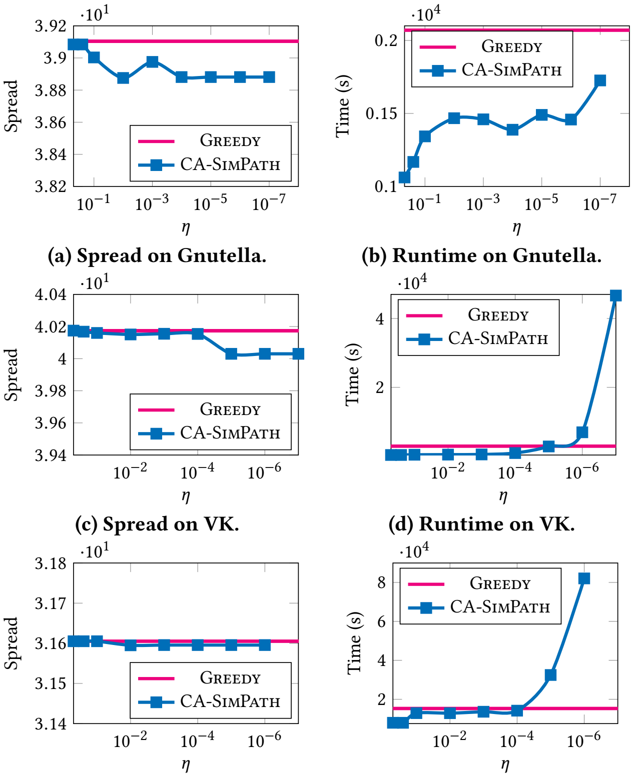
Figure 10: Performance on Citation vs. seed set size, $\eta = 0.05$.

7.5 Results on Citation

Figures 9 and 10 present our results on the Citation dataset; we set $\eta = 0.05$ and, as discussed in Section 6.4, $\theta = 5 \cdot 10^{-5}$, which maximizes the AUC value by the analysis of Section 5.4). Results are consistent to those with other datasets, corroborating that CA-SimPath achieves superior scalability without an efficacy drawack.

7.6 Influence of η threshold

Lastly, we study the effect of the path probability pruning threshold η on Gnutella, VK, and Citiation data. Figure 11 presents our results on an inverted x-axis. As expected, smaller threshold values incur reduced runtime, since they cause earlier pruning of path exploration, hence. Surprisingly, reducing η does not significantly affect the achieved spread values, which converge as η falls. This result indicates that CA-SimPath not only achieves results practically indistinguishable from those of Greedy, as we have seen in previous results, but also does so in a robust manner, insensitive to variations in the aggressiveness of pruning. The main parameters the affect performance appear to be network structure and parameters, while the accuracy of path probability estimation is secondary.



(a) Spread on Gnutella. **(b) Runtime on Gnutella.**

(c) Spread on VK. **(d) Runtime on VK.**

(e) Spread on Citation. **(f) Runtime on Citation.**

Figure 11: Performance of CA-SimPath vs. η.

8 CONCLUSION

We proposed a *content-aware* variation of the linear threshold model that governs influence spread over a network based on features associated with both network nodes and propagated items. We formulated the problem of *influence maximization by feature selection* under this model, and proposed a solution therefor, CA-SimPath, that expands on previous work. We also introduced a content-aware variation of the credit assignment algorithm to derive model parameters. We evaluated the performance of the solutions on synthetic power-law networks and real-world datasets, demonstrating that CA-SimPath significantly outperforms a Greedy baseline in terms of runtime, while preserving its effectiveness; we also showed that content-aware credit assignment achieves good predictive power.

In the future, we intend to study how to contain a viral epidemic by engineering features affecting its spread. We also plan to study the robustness of solutions to uncertain input [27], as well as the problem of learning model parameters in online fashion [19], the possibility of learning from anonymized network data [29, 36], and that of configuring learned responses adaptively while safeguarding the privacy of exchanged user information, in the spirit of [15].

REFERENCES

[1] Sinan Aral and Dylan Walker. 2011. Creating Social Contagion Through Viral Product Design: A Randomized Trial of Peer Influence in Networks. *Management Science* 57, 9 (2011), 1623–1639.

[2] Nicola Barbieri and Francesco Bonchi. 2014. Influence Maximization with Viral Product Design. In *SIAM SDM*.

[3] Nicola Barbieri, Francesco Bonchi, and Giuseppe Manco. 2012. Topic-Aware Social Influence Propagation Models. In *ICDM*.

[4] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent Dirichlet Allocation. *J. Mach. Learn. Res.* 3 (2003), 993–1022.

[5] Christian Borgs, Michael Brautbar, Jennifer T. Chayes, and Brendan Lucier. 2014. Maximizing Social Influence in Nearly Optimal Time. In *SODA*.

[6] Wei Chen, Chi Wang, and Yajun Wang. 2010. Scalable Influence Maximization for Prevalent Viral Marketing in Large-Scale Social Networks. In *KDD*.

[7] Yi-Cheng Chen, Wen-Chih Peng, and Suh-Yin Lee. 2012. Efficient Algorithms for Influence Maximization in Social Networks. *Knowl. Inf. Syst.* 33, 3 (2012), 577–601.

[8] Irena Pletikosa Cvijikj and Florian Michahelles. 2013. Online engagement factors on Facebook brand pages. *Social Netw. Analys. Mining* 3, 4 (2013), 843–861.

[9] Lisette de Vries, Sonja Gensler, and Peter S.H. Leeflang. 2012. Popularity of Brand Posts on Brand Fan Pages: An Investigation of the Effects of Social Media Marketing. *Journal of Interactive Marketing* 26, 2 (2012), 83–91.

[10] Amit Goyal, Francesco Bonchi, and Laks V. S. Lakshmanan. 2010. Learning Influence Probabilities in Social Networks. In *WSDM*.

[11] Amit Goyal, Francesco Bonchi, and Laks V. S. Lakshmanan. 2011. A Data-Based Approach to Social Influence Maximization. *Proc. VLDB Endow.* 5, 1 (2011), 73–84.

[12] Amit Goyal, Wei Lu, and Laks V. S. Lakshmanan. 2011. SIMPATH: An Efficient Algorithm for Influence Maximization under the Linear Threshold Model. In *ICDM*.

[13] Petter Holme and Beom Jun Kim. 2002. Growing scale-free networks with tunable clustering. *Phys. Rev. E* 65 (2002), 026107. Issue 2.

[14] Sergei Ivanov, Konstantinos Theocharidis, Manolis Terrovitis, and Panagiotis Karras. 2017. Content Recommendation for Viral Social Influence. In *SIGIR*.

[15] Panagiotis Karras, Artyom Nikitin, Muhammad Saad, Rudrika Bhatt, Denis Antyukhov, and Stratos Idreos. 2016. Adaptive Indexing over Encrypted Numeric Data. In *SIGMOD*.

[16] Elihu Katz. 1959. Mass communications research and the study of popular culture: An editorial note on a possible future of this journal. *Studies in Public Communication* 2 (1959), 1–6.

[17] David Kempe, Jon Kleinberg, and Éva Tardos. 2003. Maximizing the Spread of Influence through a Social Network. In *KDD*.

[18] Ricardo Lage, Peter Dolog, and Martin Leginus. 2013. Vector Space Models for the Classification of Short Messages on Social Network Services. In *WEBIST*.

[19] Siyu Lei, Silviu Maniu, Luyi Mo, Reynold Cheng, and Pierre Senellart. 2015. Online Influence Maximization. In *KDD*.

[20] Jure Leskovec, Jon M. Kleinberg, and Christos Faloutsos. 2005. Graphs over Time: Densification Laws, Shrinking Diameters and Possible Explanations. In *KDD*.

[21] Jianxin Li, Chengfei Liu, Lu Chen, Zhenying He, Amitava Datta, and Feng Xia. 2017. iTopic: Influential Topic Discovery from Information Networks via Keyword Query. In *WWW Companion*.

[22] Jianxin Li, Chengfei Liu, Jeffrey Xu Yu, Yi Chen, Timos K. Sellis, and J. Shane Culpepper. 2016. Personalized Influential Topic Search via Social Network Summarization. *IEEE TKDE* 28, 7 (2016), 1820–1834.

[23] Yuchen Li, Ju Fan, George V. Ovchinnikov, and Panagiotis Karras. 2019. Maximizing Multifaceted Network Influence. In *ICDE*.

[24] Yuchen Li, Ju Fan, Yanhao Wang, and Kian-Lee Tan. 2018. Influence Maximization on Social Graphs: A Survey. *IEEE TKDE* 30, 10 (2018), 1852–1872.

[25] Alvis Logins and Panagiotis Karras. 2019. Content-based Network Influence Probabilities: Extraction and Application. In *ICDM Workshops*.

[26] Alvis Logins and Panagiotis Karras. 2019. An Experimental Study on Network Immunization. In *EDBT*.

[27] Alvis Logins, Yuchen Li, and Panagiotis Karras. 2020. On the Robustness of Cascade Diffusion under Node Attacks. In *The Web Conference*.

[28] George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. 1978. An Analysis of Approximations for Maximizing Submodular Set Functions–I. *Math. Program.* 14, 1 (1978), 265–294.

[29] Sadegh Nobari, Panagiotis Karras, HweeHwa Pang, and Stéphane Bressan. 2014. L-opacity: Linkage-Aware Graph Anonymization. In *EDBT*.

[30] Matei Ripeanu, Adriana Iamnitchi, and Ian T. Foster. 2002. Mapping the Gnutella Network. *IEEE Internet Comput.* 6, 1 (2002), 50–57.

[31] Peter J. Rousseeuw. 1987. Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis. *J. Comput. Appl. Math.* 20, 1 (1987), 53–65.

[32] Kazumi Saito, Ryohei Nakano, and Masahiro Kimura. 2008. Prediction of Information Diffusion Probabilities for Independent Cascade Model. In *Knowledge-Based Intelligent Information and Engineering Systems*.

[33] Youze Tang, Xiaokui Xiao, and Yanchen Shi. 2014. Influence Maximization: Near-Optimal Time Complexity Meets Practical Efficiency. In *SIGMOD*.

[34] Guangmo Tong, Ruiqi Wang, Zheng Dong, and Xiang Li. 2020. Time-constrained Adaptive Influence Maximization. (2020). arXiv:2001.01742v2

[35] Leslie G. Valiant. 1979. The Complexity of Enumeration and Reliability Problems. *SIAM J. Comput.* 8, 3 (1979), 410–421.

[36] Mingqiang Xue, Panagiotis Karras, Chedy Raïssi, Panos Kalnis, and Hung Keng Pung. 2012. Delineating Social Network Data Anonymization via Random Edge Perturbation. In *CIKM*.