Content Aware Influence Maximization on the LT Model

ABSTRACT

(AK: Highly inspired from CRVSI, reframe to avoid verbatim copy)

Influence maximization - the problem of maximizing the spread of a message, advertisement or a campaign has been an important problem for various practical reasons. How does one decide the content which will become viral in a whole network after we share it with friends or followers? For an online campaign, how does one decide the groups on which the meme has to be posted. The latter 2 questions can be abstracted out to the selection of seeds or 'early adopters' in the social network **(AK: insert citations)**. The problem of selecting the seed set for social influence maximization is not new and has been studied extensively **(AK: insert citations)**. Yet, in many real-world settings, the opposite holds: a meme's propagation depends on users' perceptions of its *tunable* characteristics, while the set of initiators is *fixed*.

Instead of focusing on choosing the initial set of adopters, we focus on choosing the content attributes for maximizing the spread of a meme across the network. To our knowledge, no previous works except [4] studies the choice of content features for spread maximization. We expand upon [4] and generalize feature selection to the linear threshold model. We prove that the spread maximization problem in the choice of features is not submodular as compared to the seed selection problem [5]. Thus, we diverge from the greedy algorithm and propose a new beam search based algorithm for feature selection. We further show applications of the beam search based algorithms for a variety of situations in influence maximization.

ACM Reference format:

. 2020. Content Aware Influence Maximization on the LT Model. In *Proceedings of The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, , 2019 (KDD '20)*, 8 pages.

https://doi.org/

1 INTRODUCTION

An idea or innovation can spread across a social network. Understand - how ideas are "adopted". Goal - to achieve the max possible spread across a network. Optimize over content features. Describe linear threshold model and cite a paper that corresponds to showing comparison b/w IC and LTM. Justify adaptation to LTM. **(AK: Need to tell about social influence and it's applications. Again, same as CRVSI paper?)**

KDD '20, 2019,

ACM ISBN .

https://doi.org/

2 MOTIVATION AND BACKGROUND

(AK: Removed the section for clarity, will add later)

We start our discussion drawing from cognitive science and marketing, paving the way for our problem definition. **(AK: Emphasis on choice of the Linear Threshold Model. Practical situations where the model makes more sense than the IC model. Previous works focus on seed set, but we focus on content recommendation given a network structure.)**

3 PROBLEM STATEMENT

3.1 Notations

Let $F = \{f_1, \ldots, f_k\}$ be a set of features (topics) of actions (propagated items, messages). The Content-Aware LT (CLT) model defines the activation probability of a node v as

$$P(a|v) = \sum_u p_{uv} + q_{uv}(|F_v \cap F_a|) \geq \theta_v$$

where u is in-neighbour of v; p_{uv}, q_{uv} are labels of the edge (u, v) that indicate the influence of u on v; F_v is a set of topics the node v is interested in, F_a is a set of topics that characterize the action i; and $\theta_v \in [0, 1]$ is a threshold that is randomly chosen by the node v in the beginning of propagation.

The parameters of the model are p_{uv}, q_{uv}, F_v and F_i, where the last two are discrete sets that need to be determined.

Let $t(a)$ be a timestamp of the action, and let us call an action a performed by v as *adapted* by u if both have performed the action sequentially in time

$$t_v(a) < t_u(a)$$

3.2 Proof for non-sub/super-modularity for the Linear Threshold Model

(AK: Need to update figures with correct labels on the edges)

It turns out that the spread function $\sigma(F)$ is neither submodular or supermodular under the assumptions that the influence weights (which takes into account the features of the content and the user's preferences) are (i) monotonic and (ii) submodular (?). The "feature-inclusive influence weight" $b^F_{u,v}$ for an edge (u, v) is said to be monotonic on F if it holds that for subsets of attributes $F_1 \subset F_2 \subset \Phi$; $b^{F_1}_{u,v} \leq b^{F_2}_{u,v}$. The spread of a feature set F can be calculated using the feature-inclusive weights $b^F_{u,v}$ using the following formula [2] which can be derived using the "live-edge" model [5]:

$$\sigma(S) = \sum_{v \in V} \sum_X \Pr[X] \cdot I(S, v, X) = \sum_{v \in V} \Upsilon_{S,v}$$

Where $\Upsilon_{S,v}$ is the probability that v activates if S is chosen as the intial seed set. Let $P = \langle v_1, v_2, \ldots v_m \rangle$ be a simple path. A simple path is a path in which no nodes are repeated. Clearly, $\Pr[P] = \prod_{(v_i, v_j) \in P} b_{v_i, v_j}$. By definition of the "live-edge" [5] model we have [1, 2],

$$\Upsilon_{u,v} = \sum_{P \in \mathcal{P}(u,v)} \Pr[P]$$

where Pr[P] is the probability of a path P being live and $\mathcal{P}(u, v)$ is set of all paths from node u to v.

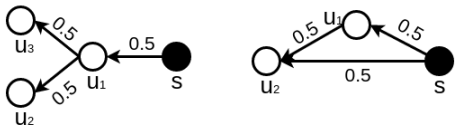

Figure 1: Increasing & decreasing marginal returns

We now examine whether the spread function $\sigma(F)$ is a submodular function. It turns out that the function is not submodular, which we prove using the following counter examples:

Why is this claim here, and where is the proof? **(AK: Replacing claim 1 with explicit example)**

Example 1: In this example we show that the spread function ($\sigma(F)$) has increasing marginal gain upon adding a feature f to the subsets of features F_1, F_2 s.t. $F_1 \subset F_2 \subset \Phi$.

For F_1

$$b_{sv_1}^{F_1} = 1/2, b_{sv_2}^{F_1} = b_{v_1v_2}^{F_1} = 1/2$$
$$\sigma(F_1) = \Upsilon_{s,v_1} + \Upsilon_{s,v_2}$$
$$\sigma(F_1) = 1/2 + 2 * (1/4) = 1$$

Now, we add x to set F_1 to get the set $F_1 \cup \{x\} = \{x\}$ and calculate $\sigma(F_1 \cup \{x\})$

$$b_{sv_1}^{F_1\cup\{x\}} = 1, b_{sv_2}^{F_1\cup\{x\}} = b_{v_1v_2}^{F_1\cup\{x\}} = 2/3$$
$$\sigma(F_1 \cup \{x\}) = \Upsilon_{s,v_1} + \Upsilon_{s,v_2}$$
$$\sigma(F_1 \cup \{x\}) = 1 + 2 * (2/3) = 7/3$$

Thus,

$$\Delta_1 = 7/3 - 1 = 4/3$$

For F_2

$$b_{sv_1}^{F_2} = 1/2, b_{sv_2}^{F_2} = b_{v_1v_2}^{F_2} = 5/6$$
$$\sigma(F_2) = \Upsilon_{s,v_1} + \Upsilon_{s,v_2}$$
$$\sigma(F_2) = 1/2 + 2 * 5/12 = 4/3$$

Now, we add x to set F_2 to obtain the set $F_2 \cup \{x\} = \{x, y\}$ and calculate $\sigma(F_2 \cup \{x\})$

$$b_{sv_1}^{F_2\cup\{x\}} = 1, b_{sv_2}^{F_2\cup\{x\}} = b_{v_1v_2}^{F_2\cup\{x\}} = 1$$
$$\sigma(F_2 \cup \{x\}) = \Upsilon_{s,v_1} + \Upsilon_{s,v_2}$$
$$\sigma(F_2 \cup \{x\}) = 1 + 2 * 1 = 3$$

Thus,

$$\Delta_2 = 3 - 4/3 = 5/3$$

Since $\Delta_2 > \Delta_1$, we conclude that the spread function is not submodular.

Example 2: In this example, we demonstrate a case of decreasing marginal gains for 2 subsets of features s.t. $F_1 \subset F_2 \subset \Phi$. Consider features $x, y \in \Phi$. Consider $F_1 = \phi(NULL)$ and $F_2 = y$.

You need to define all notations and terms used below!

For F_1

$$b_{sv_1}^{F_1} = 1/2, b_{sv_2}^{F_1} = b_{v_1v_2}^{F_1} = 1/4$$
$$\sigma(F_1) = \Upsilon_{s,v_1} + \Upsilon_{s,v_2}$$
$$\sigma(F_1) = 1/2 + [1/4 + 1/2 * 1/4] = 7/8$$

Now, we add x to set F_1 to get the set $F_1 \cup \{x\} = \{x\}$ and calculate $\sigma(F_1 \cup \{x\})$

$$b_{sv_1}^{F_1\cup\{x\}} = 3/4, b_{sv_2}^{F_1\cup\{x\}} = b_{v_1v_2}^{F_1\cup\{x\}} = 1/2$$
$$\sigma(F_1 \cup \{x\}) = \Upsilon_{s,v_1} + \Upsilon_{s,v_2}$$
$$\sigma(F_1 \cup \{x\}) = 3/4 + [1/2 + 3/4 * 1/2] = 13/8$$

Thus,

$$\Delta_1 = 13/8 - 7/8 = 0.75$$

For F_2

$$b_{sv_1}^{F_2} = 3/4, b_{sv_2}^{F_2} = b_{v_1v_2}^{F_2} = 1/4$$
$$\sigma(F_2) = \Upsilon_{s,v_1} + \Upsilon_{s,v_2}$$
$$\sigma(F_2) = 3/4 + [1/4 + 3/4 * 1/4] = 19/16$$

Now, we add x to set F_2 to obtain the set $F_2 \cup \{x\} = \{x, y\}$ and calculate $\sigma(F_2 \cup \{x\})$

$$b_{sv_1}^{F_2\cup\{x\}} = 4/5, b_{sv_2}^{F_2\cup\{x\}} = b_{v_1v_2}^{F_2\cup\{x\}} = 1/2$$
$$\sigma(F_2 \cup \{x\}) = \Upsilon_{s,v_1} + \Upsilon_{s,v_2}$$
$$\sigma(F_2 \cup \{x\}) = 4/5 + [1/2 + 4/5 * 1/2] = 17/10$$

Thus,

$$\Delta_2 = 17/10 - 19/16 = 0.5125$$

Since $\Delta_2 < \Delta_1$, we conclude that the spread function is not supermodular.

4 PREVIOUS WORK

4.1 CRVSI

Idea for content based feature recommendation.

4.2 SIMPATH

Explanation and intuition of the algorithm, particulary the formulas and results being used for optimizations.

5 LINEAR THRESHOLD MODEL

(AK: FIX: formula now incorporates the weights using in-degrees instead of 1/Fv)
(AK: FIX2: data augmentation)

———————————————

5.1 Notation

Edge weights from dataset	$b_{u,v}$
Q factor	$q_{u,v}$
Transformed edge weights	b^t
in-degree of vertex v	d_{in}^v
Feature inclusive weights	$f_{u,v}$
Feature set of the content	F
Feature set of of vertex v	F_v
Intersection of F and F_v	h
Sigmoid function	$\sigma()$

Table 1: Summary of the notation

5.2 Data Augmentation

(AK: Since SIMPATH was based on LTM, it makes sense that we use their data sets for our experiments since the values will be tailored accordingly.) Using data sets meant to be for the IC model, we adjust the weight values to get the weights into a compatible range. The following augmentation is applied:

$$b^{aug}_{u,v} = k * min(b_{u,v}, d^v_{in})$$

With tunable hyperparameter k. From now on weights from the augmented data ($b^{aug}_{u,v}$) will be simply referred to as $b_{u,v}$.

5.3 Derivation for weight formulae

The feature inclusive weight design problem is slightly more complex for the LT Model as compared to the IC Model. One must ensure that sum of all incoming weights for a vertex remains ≤ 1 irrespective of the selected features.

For simplicity, we choose the weights to be weighted sigmoids, such that the sum of the weights remain ≤ 1. Further, the edge weight function is supposed to be monotonic.

Further, the dataset gives values for edge weights. These cannot be plugged directly into the sigmoid function. We apply a transform to the weight values to make them compatible for using inside the sigmoid function. Let these transformed values be b^t.

$$f_{u,v} = 1/d^v_{in} * \sigma(b^t_{u,v} + q_{u,v} * h)$$

The $1/d_in$ factor ensures that th We obtain closed form expression for the edge weights $f_{u,v}$ by solving the following equations:

- The feature inclusive weights are equal to the input weights given by the (augmented) dataset

$$F = \emptyset; f_{u,v} = b_{u,v}$$

- The feature inclusive weights grow to c times the input weights upon inclusion of all the groups that a vertex belongs to.

$$F = F_v; f_{u,v} = c * b_{u,v}$$

The results from the above equation are summarized in the next subsection. Please refer to table 1 for a summary of the notation used.

5.4 Forumlae for weights

$$\text{base weights for edge } (u,v) = b_{uv}$$
$$\text{(tuning factor) c } = 5$$
$$\text{(q factor) } q_{uv} = 1/|F_v| * log(\frac{(c - cb_{uv}d^v_{in})}{(1 - c * b_{uv}d^v_{in})})$$
$$\text{Feature set of propagated meme } = F$$
$$\text{Feature set of in-node } (v) = F_v$$
$$\text{in-degree } = d^v_{in}$$
$$\text{Transformed base weights } b^t_{uv} = log(b_{uv}d^v_{in}/(1 - b_{uv}d^v_{in})))$$
$$f\_weights = 1/d^v_{in} * \sigma(b^t_{uv} + q_{uv}|F_v \cap F|)$$

where σ represents the sigmoid function. Note that since $\sigma(z)$ is ≤ 1 the sum of all edge weights (for incoming edges) for a vertex v remains ≤ 1. **(AK: set to inf (sigmoid = 1) whenever negative value inside log.)**

5.4.1 Greedy. Implemented MC simulations for the LT model. Reference, SIMPATH comparison code. Using same parameters. Greedy overall architecture remains the same as the explore update greedy code. adv - modify calculation of spread with incorporation to the LT model.

Algorithm 1: Greedy(G, S, k)

```
F = ∅ while |F| < k do
    for every f ∈ Φ \ F do
        calculate σ(F + {f}) using Monte Carlo simulations
    F = F ∪ arg max_f {σ(F + {f})}
return F
```

5.4.2 Optimized.

6 SOLUTION

6.1 SIMPATH based feature selection

We propose feature selection algorithm based on interesting properties explored by [2]. The feature selection algorithm maintains the same structure as that of the Greedy one, but uses a new spread estimate function which relies on spread estimation using our simpath_spread algorithm. From **??** we have, that the spread of a given seed set S is equal to the contribution to each seed (s) spread on the induced subgraph of G with vertex set V-S+s.

(AK: Theorem 1 taken from SIMPATH paper.) Theorem 1. *In the LT model, the spread of set S is the sum of the spread of each node $u \in S$ on subgrpahs induced by V - S + u. That is,*

$$\sigma(S) = \sum_{u \in S} \sigma^{V-S+u}(u)$$

Further, the contribution of each seed vertex to the spread ($\sigma^V(s)$) is given by:

$$\sigma^V(s) = \sum_{v \in V} \Upsilon_{s,v}$$

where, $\Upsilon_{s,v}$ is the probability that v activates if s belongs to the initial seed set. As defined above in the analysis section we have ...

For a given feature set F, the edge weight probabilities being fixed to P, the spread of a single seed vertex (s) can be estimated by enumerating all simple paths originating from s. Unfortunately, the problem of enumerating all simple paths is #P-hard [7].

To speed up the spread estimation, we use a pruning factor η as in [2]. This does not hurt the estimate value while producing significant savings in runtime **??**. **(AK: from SIMPATH:)** This hold true since majority of the influence can be captured by exploring

We maintain data structures D and Q. D maintains the set of vertices that have been

(AK: Come up with interesting names of the algorithm)

Algorithm 2: simpath(G, S, Φ)

```
declare: selected = ∅ spread, max_feature, max_spread for f ∈ Φ - selected do
    max_feature = argmax(simpath_spread(G, F+f, S)) selected = selected ∪ f
```

[[[AL: 2 algorithms commented out. they compile with errors because they are missing $]]]

OPTIMIZATIONS to discuss:-

(1) in_edges being cleared out of all the vertices in the seed set
(2) the faster spread calculation algorithm based on enumerating simple paths pruned by η
(3) Limiting the feature selection loop to only high-impact features - those which led to pruned paths **(AK: need to give substantial argument - just another heuristic as compared to MIP - should introduce as a variant like a 'Plus' version)**

6.2 BEAM SEARCH ALGORITHM

To improve the quality of the feature set. **[[[AL: algorithm commented out here. it compiles with errors due to missing $]]]**

6.3 Beam search in the Linear Threshold Model

Beam search, a common algorithm in finding the best optimal sequence **(AK: cite beam search intro paper)** used extensively in problems such as Automatic Speech Recognition (HMM state tagging), PoS tagging

7 DATASET

7.1 Seed Selection

The goal of the content aware influence maximization is to select a feature set for *any* seed set. However, we study the problem when seeds are selected according to simple influence maximization heuristics, as seeds are usually selected so to archieve a large spread. A list of seed selection strategies to consider:

- Random selection (naïve baseline)
- Degree centrality (top nodes with largest degrees)
- Betweeness centrality
- PageRank centrality

We consider seed sets of sizes 1, 10, 100.

7.2 Synthetic Dataset

As a synthetic network, we use **Barabási-Albert** (BA) networks. They have both high clustering coefficients and power-law degree distribution, hence are better imitations of real-world social networks. We use the algorithm of Holme and Kim [3], which extends the original Barabási-Albert model, yet use the BA label as its basis; this algorithm randomly creates m edges for each node in a graph, and for created edge with a probability p adds an edge to one of its neighbors, thus creating a triangle. Suggested parameter values: $p = 0.2$, $m = 15$. **[[[AL: generator for this dataset can be found by the link in the comments]]]**

We generate parameters of a dataset according to a trivalency model. For each edge, we uniformly in random pick one of three values per parameter:

$$p_{uv} \in \{0.1, 0.5, 0.9\}$$

$$q_{uv} \in \{0.1, 0.5, 0.9\}$$

We set a size of a feature vector to 100

$$|F| = 100$$

and we randomly set elements of this vector to 1 with probability p^+, per each F_v and F_a. We assume sparse feature vectors, therefore we set $p^+ = 0.1$. **[[[AL: all parameters might be selected differently, I suggest them arbitrarily]]]**

We study networks of sizes $2^9, ..., 2^{14}$.

7.3 Real-World Dataset

7.3.1 VK data. We use the VK dataset, presented by Logins et al [6]. As a result of Natural Language Processing, each message of a user is associated with a vector of topics. In our problem, we require a binary vector of topics $t = \{t_1..t_k\}$ per message and per user. To derive a binary vector, we introduce a threshold parameter θ, so that

$$F = \{f_1, .., f_k\} = \{I(t_1 > \theta), I(t_2 > \theta), ...\}$$

where $I(\cdot)$ is an indicator function. Let us denote $I(t)$ as a vector F resulted from applyting the indicator function per element.

We also define a vector of user preferences (interests) as a thresholded average over all messages authored by the user:

$$F_u = I\left(\frac{1}{|T|}\sum_T t_i\right)$$

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

Since k resulting from the NLP analysis is quite large ($k = 6.6 \cdot 10^4$), we reduce k by applying the Principle Component Analysis (PCA) dimensionality reduction on a sample of topics in order to get the dimensionality of $k = 1000$.

Messages that result with the same topic distribution after applying the threshold are considered the same action.

7.3.2 Threshold Selection. We select θ such that it maximizes the Area Under ROC Curve. Following [6], for each action m and each neighbor of its author, we consider the existence of the same action post m' on the neighbor's wall as a *positive* instance of propagation. Scanning the log of posts, we derive True Positive and False Positive Rates for all values of θ, TPR = $\frac{TP}{TP+FN}$, FPR = $\frac{FP}{FP+TN}$. We evaluate the quality of the trained probability model by the Area Under Curve: AUC = $\int$ TPR dFPR. We select θ such that it maximizes *AUC*.

7.4 Learning parameters

7.4.1 The Credit Assignment in the LT model. The algorithm is based on the assumption that *for each **successful** activation, all in-neighbours share the same "credit" for that activation.* For example, if two nodes have posted a message about Trump, and then their common neighbour also posted about Trump, then former two nodes are both 50% responsible for activating the third node. *It does not matter whether the ground-truth activation **probabilities** might be different from each other*, because nodes share credits only for successful actions.

For the regular LT model, the credit assignment algorithm determines influence probabilities as

$$p_{uv} = \frac{\sum_a credit_{uv}(a)}{A_v}$$

where a is an action that propagates in one diffusion instance (cascade). The credit is defined as

$$credit_{uv}(a) = \frac{1}{\sum_{w \in S} I(t_w(a) < t_u(a))}$$

where I is an indicator that an in-neighbour w of a node u has been activated with an action a earlier than u.

7.4.2 The Credit Assignment in the CLT model. First, we must assume that topics of actions and user preferences are predefined, i.e. must be determined *before* applying the credit assignment algorithm. This is a crucial difference to the EM algorithm.

Second, for the CLT model, we still use the same *equal credit share* assumption. That means, for each successful action an in-neighbour takes $\frac{1}{d}$ credit, where d is the total number of in-neighbours that share credit for that action.

However, instead of a single value p_{uv}, we should find coefficients p_{uv} and q_{uv}, and these coefficients are dependent on the topics of each action. We will use the *Ordinary Least Squares* (OLS) estimator. Let S be a set of in-neighbours of v. Let A_u be a set of all action performed by u, and A_{uv} be a set of actions performed by u and adapted by v. Then, for each $a \in A_u$, the equal share assumption implies that

$$p_{uv} + q_{uv}\alpha_v(a) = \mathbb{E}_a \left[\frac{I(t_u(a) < t_v(a))}{\sum_{w \in S} I(t_w(a) < t_v(a))} \right] \tag{1}$$

where the indicator function $I(t_u(a) < t_v(a))$ shows that v adopted action a at any time after u, $\sum_{w \in S} I(t_w(a) < t_v(a))$ shows the number of in-neighbours who could have influenced v if v has ever adopted a, and $\alpha_v(a) = |F_v \cap F_a|$ is the coefficient that shows how much topics of a are similar to preferences of v.

Let $d_v(a) = \sum_{w \in S} I(t_w(a) < t_v(a))$, and $n_u = |A_u|$. Then, OLS is given by

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

$$p_{uv} = \frac{1}{n_u} \sum_{a \in A_{uv}} \frac{1}{d_v(a)} - q_{uv}\frac{1}{n_u} \sum_{a \in A_u} \alpha_v(a) \tag{3}$$

The equations above come from a simple OLS for a statistics of a form

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

where

$$y_i = \frac{I(t_u(a) < t_v(a))}{\sum_{w \in S} I(t_w(a) < t_v(a))}$$
$$x_i = \alpha_v(a)$$
$$\beta = q_{uv}$$
$$\alpha = p_u v$$

,and ϵ_i is random errors.

7.4.3 The algorithm. Goyal et al. proposed an algorithm that calculates all necessary values for calculating p_{uv} in a minimal number of scans through data. We adapt their algorithm. In order to compute coefficients of CLT according to Eq. 3 and 2, we have to know the following statistics:

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

Then, coefficients are equal to

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

$$p_{uv} = \frac{1}{n_u}C^3_{uv} - q_{uv}\frac{1}{n_u}C^2_{uv} \tag{5}$$

Let E be a set of edges of a graph. Logs of actions should be sorted in chronological order.

7.4.4 The Difference between actions and posts. When working with text messages, each post is assotiated with a feature vector. In order to define *cascades* over the data, we have to define when two posts of neighbour nodes can count as an instance of a successful influence. There are 3 conditions for that:

- Two posts are subsequent in time
- The time difference between messages are sufficiently small
- Messages have similar content

We define messages that have similar content as messages with the same binary topic distribution F. We note here that *an action* a used in the equations in former sections is *a set of all messages with the same topic distributions*. For example, if u and v posted 2 messages of distributions F_1 and F_2, and $F_1 = F_2$, then it counts as performing *the same action* a. If $F_1 \neq F_2$, then it counts as two different actions a_1 and a_2.

We also note that we assume each node can be influenced on a topic only once, but may post about the same topic several times. Therefore, for each node v and action a we set a time step of that action performed by v

- as the time step of the *earliest* message with F_a, when v is the *target* node
- a time step of *any* message with F_a, when v is the *source* node

As a result, each action a is associated with a *set* of time steps.

If we want to enforce a time limit within which a node can be influenced, then we should force the requiremet for two messages to be the same successful action only if $0 < t_v - t_u < \tau$, where τ is a time threshold, instead of messages being subsequent only $(t_v > t_u)$.

7.4.5 The Algorithm. Now we will describe the Credit Assignment Algorithm for the CLT model (Algorithm 3).

First, we initiate variables. Then, for each action a we initiate an empty *currentTable* data structure, and traverse logs in chronological order considering each *message* m that represent the action a. If a message and a user already exist in *currentTable* (with any time step t), then we just update the time step in the table to t_v. The intuition behind this operation is that any other node that the

Algorithm 3: Credit Assignment

Input: A set of messages grouped as actions, sorted by t
Output: $C_{uv}^{1,2,3,4}, n_u$ for all u and v

for $u, v \in V$ **do**
  $n_v \leftarrow 0, C_{uv}^{1,2,3,4} \leftarrow 0$;
for $a \in$ *actions* **do**
  currentTable $\leftarrow \emptyset$;
  for *each* $< v, m, t_v > | m \in a$ *in chronological order* **do**
    if $< v, a, t > \in$ *currentTable* **then**
      $t \leftarrow t_v$;
    else
      Increment n_v;
      for $w \in V | (v, w) \in E$ **do**
        Compute $\alpha_w(a)$ and store in cache;
        $C_{vw}^2 += \alpha_w(a)$;
        $C_{vw}^4 += \alpha_w^2(a)$;
      parents $\leftarrow \emptyset$, $d_v(a) \leftarrow 0$;
      for $< u, a, t_u > \in$ *currentTable* $\wedge (u, v) \in E$ **do**
        if $\tau > t_v - t_u > 0$ **then**
          Increment $d_v(a)$;
          Insert u in *parents*;
      for $u \in$ *parents* **do**
        $C_{uv}^1 += \alpha_v(a) \frac{1}{d_v(a)}$;
        $C_{uv}^3 += \frac{1}{d_v(a)}$;
      Add $< v, a, t_v >$ to *currentTable*;

algorithm considers in future iterations are potential *target* nodes, so the current node v is a potential *source node*. Therefore, its time step is benefitial to update to the latest possible.

If v, a is not in the table, then we consider m as the first instance when v performs the action a. We update n_v value, and then for each *outgoing edge* (v, w) we compute $\alpha_w(a)$. We store the computed value in cache, so that we can later use it for updating C_{uv}^1. After computing $\alpha_w(a)$, we update values of C_{vw}^2 and C_{vw}^4. These values are defines as a sum over *all* actions ever performed by a source node, so we update them as soon as any node performs any action for the first time.

In order to update C_{uv}^1 and C_{uv}^3, we need to find all actual source nodes in *currentTable* who succeded in influence of v. Whenever such u appears, we increment $d_v(a)$, and save u in a temporary list *parents*. Then, we traverse that list, and update C_{uv}^1 and C_{uv}^3 for each u who has v as a successful follower. Finally, we add $< v, a, t_v >$ in the table.

After running Algorithm 3, coefficients for each edge are updated according to Eq. 5 and 4.

8 EXPERIMENTS

8.1 Other applications of beam search in Influence Maximization Problems

Since the content recommendation problem for is not submodular, the greedy algorithm does not provide an approximation bound for the quality of the solution. It has been proved in [] that if a function $f(s)$ is submodular and monotonic ... the greedy algorithm approximates the solution within a factor of $(1 - 1/e)$ of the optimal value. **(AK: Our "main method" is based on the(some optimization of the) greedy algorithm for calculating the optimal feature set.)** We examine the quality by considering a beam search based selection instead of the greedy hill-climb search. **(AK: TO-DO [] beam search introductory paper. Insert table with comparison of quality, execution time and some essential parameters.)**

Algorithm	Model	Dataset	# Nodes	K	NO BEAM	BEAM_WIDTH = 3
Greedy	IC	Toy	100	5	19.4937	19.5421
EU	IC	Toy	100	21	6.9391	-
EU	IC	Toy	100	3	6.3695	6.2646
EU	IC	VK	7420	51	-	-

Table 2: Improvements in spread values with beam search. Set of initial seed set is 9 for the toy dataset and 20 for other cases.

8.2 SIMPATH experiments

We demonstrate our efficient approach for content feature selection through a series of experiments. The greedy feature selection algorithm based on Monte-Carlo simulations for getting an estimate of the spread for selection of the next best feature is (i)significantly slower than our approach, (ii) has negligible advantage over the SIMPATH based algorithm.

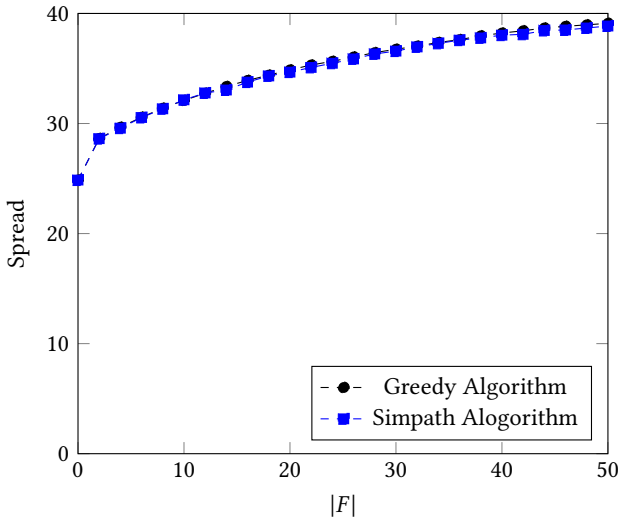

Figure 2: Spread vs the number of selected features for SIMPATH and Greedy algorithm. Eta = 1e-5. Gnutella Dataset.

In our first experiment, we demonstrate the effect of adding more content features to our meme/ad on the spread obtained. The spread values are plotted against the number of selected features. Here, the results for the greedy algorithm and the SIMPATH algorithm are in close tandem -2

[[[AL: a figure commented out here, compiles with errors]]]

Next, we show the effectiveness of the SIMPATH-based algorithm for practical purposes. (a) it could be fine-tuned for application specific scenarios where one could prioritize execution time vs

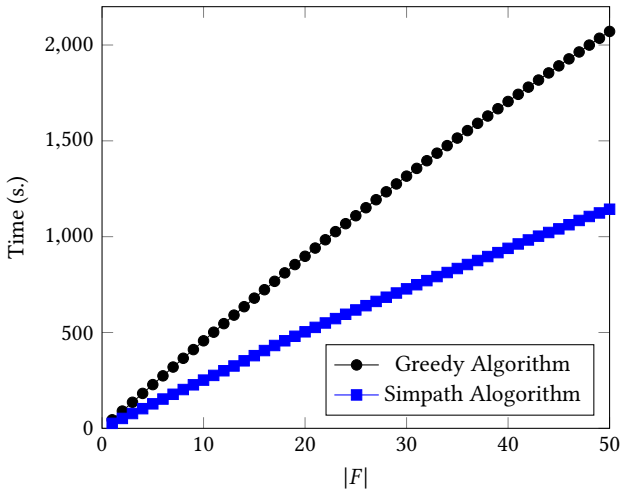

Figure 3: Execution time vs the number of selected features for the SIMPATH algorithm. Eta = 1e-5. Gnutella Dataset.

quality (b) the SIMPATH algorithm is considerably faster than the Greedy algorithm (by an order of magnitude) (c) the algorithm's execution time scales very well even with selection of much larger feature sets.

Figure -3, shows the execution time for the SIMPATH algorithm for the same parameters used in the above spread experiment. **(AK: Can get better margins with bigger eta, but for now, eta is taken to be the same as the one used in the spread experiment.)**

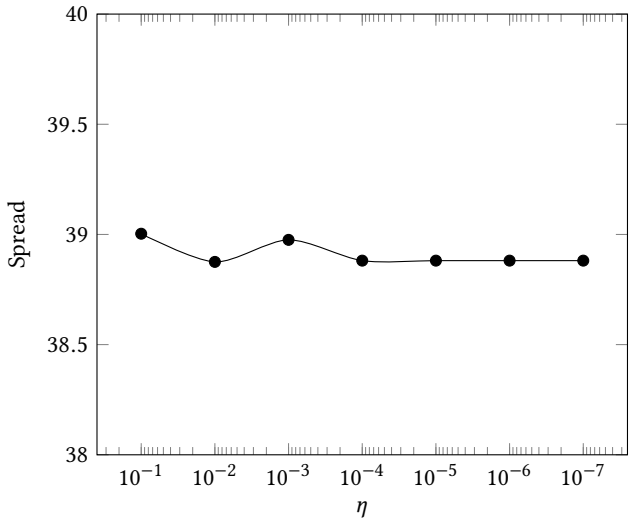

Figure 4: Spread obtained vs parameter eta for the SIMPATH algorithm. Gnutella Dataset.

To show how our algorithm can be tuned accordingly for different scenarios. We plot the execution time for the alogorithm vs the exploration parameter η in figure.

Smaller values of eta, means more exploration of the paths in the social network and hence the execution time increases with η (figure 5). Notice how the effect of having a smaller eta does not change the spread obtained by the algorithm drastically. (figure 4) This supports the motivation behind the algorithm that **(AK: influence in almost all cases spreads locally? - refer to SIMPATH paper)**

(Optimized only by the faster spread function)

Algorithm	Model	Dataset	# Nodes	K	Spread	Execution Time (s)	eta
Greedy	LTM	Toy	100	5	10.8858	422.83	-
SimPath	LTM	Toy	100	5	10.9237	633.34	1e-7
SimPath	LTM	Toy	100	5	10.9237	69.08	1e-5
Greedy	LTM	Gnutella	10876	50	39.0870	1921.95	-
SimPath	LTM	Gnutella	10876	50	38.9428	1339.11	1e-7
SimPath	LTM	Gnutella	10876	50	38.9428	1197.62	1e-5
SimPath	LTM	Gnutella	10876	50	39.03	1189 .	1e-3

Table 3: Improvements in spread values with beam search. Set of initial seed set is 9 for the toy dataset and 20 for other cases.

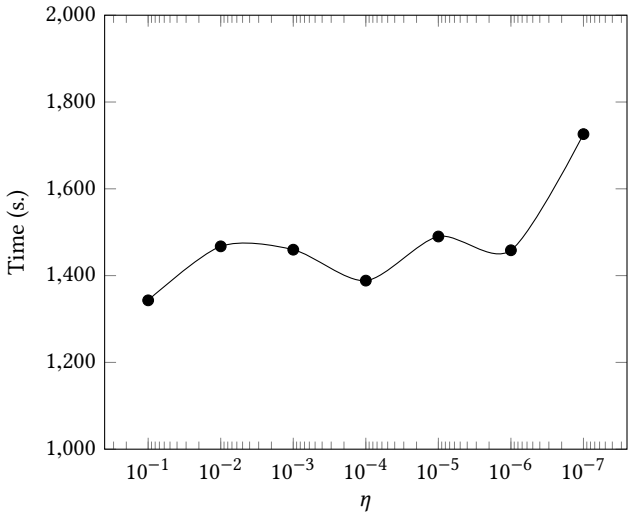

Figure 5: Execution time vs parameter eta for the SIMPATH algorithm. Gnutella Dataset.

(AK: Graphs to add x |Datasets|
1. Quality vs Eta
2. Quality vs number of selected features
3. Runtime was number of selected features
4. Runtime vs parameter eta
5. Runtime vs initial seed set size)

9 CONCLUSION AND DISCUSSION

REFERENCES

[1] W. Chen, Y. Yuan, and L. Zhang. 2010. Scalable Influence Maximization in Social Networks under the Linear Threshold Model. In *2010 IEEE International Conference on Data Mining*.
[2] A. Goyal, W. Lu, and L. V. S. Lakshmanan. 2011. SIMPATH: An Efficient Algorithm for Influence Maximization under the Linear Threshold Model. In *2011 IEEE 11th International Conference on Data Mining*.
[3] Petter Holme and Beom Jun Kim. 2002. Growing scale-free networks with tunable clustering. *Physical review E* 65, 2 (2002), 026107.

[4] Sergei Ivanov, Konstantinos Theocharidis, Manolis Terrovitis, and Panagiotis Karras. 2017. Content Recommendation for Viral Social Influence. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*.
[5] D. Kempe, J. Kleinberg, and E. Tardos. 2003. Maximizing the spread of influence through a social network. In *KDD*.
[6] Alvis Logins and Panagiotis Karras. 2019. Content-based Network Influence Probabilities: Extraction and Application. In *2019 International Conference on Data Mining Workshops (ICDMW)*. IEEE, IEEE Xplore, 69–72. https://doi.org/10.1109/ICDMW.2019.00020
[7] Leslie G. Valiant. 1979. The Complexity of Enumeration and Reliability Problems. *SIAM J. Comput.* 8 (1979).