Influence Maximization in Social Networks: Towards an Optimal Algorithmic Solution

Christian Borgs* Michael Brautbar† Jennifer Chayes‡ Brendan Lucier§

December 13, 2012

Abstract

Diffusion is a fundamental graph process, underpinning such phenomena as epidemic disease contagion and the spread of innovation by word-of-mouth. We address the algorithmic problem of finding a set of k initial seed nodes in a network so that the expected size of the resulting cascade is maximized, under the standard *independent cascade* model of network diffusion. The promise of such an algorithm lies in applications to viral marketing. However, runtime is of critical importance in this endeavor due to the massive size and volatility of the relevant networks.

Our main result is an algorithm for the influence maximization problem that obtains the near-optimal approximation factor of $(1 - \frac{1}{e} - \epsilon)$, for any $\epsilon > 0$, in time $O((m + n)\epsilon^{-3} \log n)$ where n and m are the number of vertices and edges in the network. The runtime of our algorithm is independent of the number of seeds k and improves upon the previously best-known algorithms which run in time $\Omega(mnk \cdot \text{POLY}(\epsilon^{-1}))$. Importantly, our algorithm is essentially runtime-optimal (up to a logarithmic factor) as we establish a lower bound of $\Omega(m + n)$ on the runtime required to obtain a constant approximation.

We then show how to modify our algorithm to allow a provable tradeoff between solution quality and runtime. We obtain an $\Theta(\frac{1}{\beta})$-approximation in time $O(n \cdot a(\mathcal{G}) \log^3(n)/\beta)$ for any $\beta > 1$, where $a(\mathcal{G})$ denotes the arboricity of the diffusion network $\mathcal{G}$. In particular, for graphs with bounded arboricity (as is the case for many models of network formation and empirically observed social networks) our algorithm is nearly runtime-optimal (up to logarithmic factors) for any fixed seed size k.

Our approach is based on a novel preprocessing scheme that generates a sparse hypergraph representation of the underlying network via sampling. We show that this representation makes it possible to efficiently estimate marginal influence in the original diffusion process with very few samples, and that the quality of this estimation degrades gracefully with reduced preprocessing time.

*Microsoft Research New England. Email: borgs@microsoft.com

†Department of Computer and Information Science, University of Pennsylvania. Email: brautbar@cis.upenn.edu

‡Microsoft Research New England. Email: jchayes@microsoft.com

§Microsoft Research New England. Email: brlucier@microsoft.com

1 Introduction

Diffusion is a fundamental process in the study of complex networks, modeling the spread of disease, ideas, or product adoption through a population. The common feature in each case is that local interactions between individuals can lead to epidemic outcomes. This is the idea behind word-of-mouth advertising, in which information about a product travels via links between individuals; see, for example, [3, 8, 10, 11, 19, 20, 38]. In recent years, as online social network structure has become increasingly visible, applications of diffusion models on networks to advertising have become increasingly relevant. A prominent application is a viral marketing campaign which aims to use a small number of targeted interventions to initiate cascades of influence that create a global increase in product adoption [17, 19, 24, 27].

A large part of this interest focuses on the algorithmic problem of inferring potential influencers from network topology. Given a network, how can we determine which individuals should be targeted to maximize the magnitude of a resulting cascade? [17, 24, 37]. Supposing that there is a limit k on the number of nodes to target (e.g. due to advertising budgets), the goal is to efficiently find an appropriate set of k nodes with which to "seed" a diffusion process.

In this paper we develop fast approximation algorithms for the above influence-maximization problem, under the standard *independent cascade* model of influence spread. Before describing these results, we first provide some background into the problem at hand.

The Model: Independent Cascades We adopt the *independent cascade* (IC) model of diffusion, formalized by Kempe et al. [24]. In this model we are given a directed edge-weighted graph $\mathcal{G}$ with n nodes and m edges, representing the underlying network. Influence spreads via a random process that begins at a set S of seed nodes. Each node, once infected, has a chance of subsequently infecting its neighbors: the weight of edge $e = (v, u)$ represents the probability that the process spreads along edge e from v to u. If we write $I(S)$ for the (random) number of nodes that are eventually infected by this process, then we think of the expectation of $I(S)$ as the *influence* of set S. Our optimization problem, then, is to find set S maximizing $\mathbb{E}[I(S)]$ subject to $|S| \leq k$.

For the corresponding algorithmic problem we assume that the network topology is described in the *sparse* representation of an (arbitrarily ordered) adjacency list for each vertex, as is natural for sparse graphs such as social networks.

The IC model captures the common intuition that influence can spread stochastically through a network, much like a disease [16, 19, 24]. Also, as noted by Kempe et al. [24], IC is equivalent to a *uniform linear threshold* model in which each node becomes infected after a certain weighted fraction of its neighbors are infected, given that these thresholds are drawn uniformly from the unit interval. This thresholding behavior is also a common feature in prominent classic models of influence spread [22,33]. Another important property of the IC model is its computational tractability. Indeed, Kempe et al. show that $\mathbb{E}[I(\cdot)]$ is a submodular monotone function [24], and hence the problem of maximizing $\mathbb{E}[I(\cdot)]$ can be approximated to within a factor of $(1 - \frac{1}{e} - \epsilon)$ for any $\epsilon > 0$, in polynomial time, via a greedy hill-climbing method. In contrast, many other formulations of the influence maximization problem have been shown to have strong lower bounds on polynomial-time approximability [4, 12, 21, 33, 36].

The greedy approach to maximizing influence in the IC model described above takes time $O(kn)$, *given oracle access to the function* $\mathbb{E}[I(\cdot)]$. In general, however, these influence values must be computed from the underlying network topology. A common approach is to simulate the random diffusion process many times to estimate influence values for each node. This ultimately[1] leads to a total runtime[2] of $\Omega(mnk \cdot \text{POLY}(\epsilon^{-1}))$. Due to the massive size and temporal volatility of many

[1] After simple optimizations, such as reusing each simulation for multiple nodes.

[2] The best implementations appear to have running time $O(mnk \log(n) \cdot \text{POLY}(\epsilon^{-1}))$ [14], though to the best of our

network dataset instances, this does not provide a fully satisfactory solution: it is crucial for an algorithm to scale well with network size [13, 29]. This requirement has spawned a large body of work aimed at developing more efficient methods of finding influential individuals in social networks. See, for example, [13–15, 23, 26, 29, 32, 40]. However, to date, this work has focused primarily on empirical methods; no algorithm has been found to provide provable approximation guarantees in time asymptotically less than $\Theta(nmk)$.

What is the bottleneck in developing better algorithms? One issue is the difficulty of estimating the expected influence of a given vertex. As an illustrating example, consider the simple network of a star on n nodes[3] in which each edge is bidirectional and has weight $p = n^{-1/4}$. In this example the center node has expected influence $n^{3/4}$ (infecting each leaf with probability p), whereas each leaf has expected influence $\Theta(n^{1/2})$. In this example, estimating the influence of a leaf via random sampling requires significant effort: one would have to realize the graph $\Omega(\frac{1}{p})$ times to obtain a reasonable estimate. As each realization requires linear time (to provide estimates for all leaves), a superlinear runtime would be required to estimate influences even if we are willing to tolerate multiplicative errors as large as $n^{1/4-\epsilon}$. A different approach is thus required if we are to achieve running time that is close to linear in the network size.

First Result: A Quasi-Linear Time Algorithm Our first and main result is an algorithm for finding $(1 - 1/e - \epsilon)$-approximately optimal seed sets in arbitrary directed networks, which runs in time $O((m + n)\epsilon^{-3} \log n)$. Importantly, the runtime of our algorithm is independent of the number of seeds k and is essentially runtime optimal as we give a lower bound of $\Omega(m + n)$ on the runtime required to obtain a constant approximation for this problem (assuming an adjacency list representation). We also note that this approximation factor is nearly optimal, as no polytime algorithm achieves approximation $(1 - \frac{1}{e} + \epsilon)$ for any $\epsilon > 0$ unless P = NP [24, 25].

Our method is randomized, and it succeeds with probability 3/5; moreover, failure is detectable, so this success probability can be amplified through repetition.

Our algorithm proceeds in two steps. First, we apply random sampling techniques to preprocess the network and generate a sparse hypergraph representation that estimates the influence characteristics of each node. Each hypergraph edge corresponds to a set of individuals that was influenced by a randomly selected node in the *transpose* graph. This preprocessing is done once, resulting in a structure of size $O((m + n)\epsilon^{-3} \log(n))$. This hypergraph encodes our influence estimates: for a set of nodes S, the total degree of S in the hypergraph is approximately proportional to the influence of S in the original graph. We can therefore run a standard greedy algorithm on this hypergraph to return a set of size k of approximately maximal total degree.

To make this approach work one needs to overcome several inherent difficulties. First, we show that the marginal influence of a node v is proportional to the probability that v is influenced by a randomly chosen node u in the transpose graph. These probabilities can be estimated more efficiently than influence itself. In particular, this transpose-graph formulation simplifies the process of estimating *marginal* influence, so that we need not repeat the estimation procedure when considering different partial solutions. This results in substantial savings in runtime.

The next difficulty to overcome is the stringent runtime constraint — we must construct our hypergraph in time $O((m + n)\epsilon^{-3} \log(n))$. We show that this number of steps suffices to approximate the influence of each set of nodes in the graph. This approximation comes, for each set, as a probabilistic guarantee with confidence $1 - 1/\text{POLY}(n)$. Finally, in order to prevent errors from accumulating when applying the greedy algorithm to the hypergraph, it is important that our estimator for influence (i.e. total hypergraph degree) is itself a monotone submodular function.

knowledge a formal analysis of this runtime has not appeared in the literature.

[3]Of course, the star graph is simple enough that there are obvious alternative methods for finding appropriate seed sets; we present it merely to illustrate the potential runtime requirements of estimating node influences directly.

Our algorithm accesses the network structure in a very limited way, and is therefore implementable in a wide array of graph access models. The only operations used by our algorithm are accessing a random vertex and traversing the edges incident to a previously-accessed vertex. In particular, our algorithm falls within the jump and crawl paradigm of [7].

Second Result: A Sublinear Time Algorithm We extend our approximation algorithm to allow a provable tradeoff between runtime and approximation quality. Given a network G with arboricity[4] $a(G)$ and a parameter $\beta \in (1, n]$, our algorithm attains approximation $\Theta(\frac{1}{\beta})$ in time $O(\frac{n \log^3(n) \cdot a(G)}{\beta})$. To the best of our knowledge, ours is the first algorithm with such a tradeoff between runtime and approximation quality.

In particular, on networks of bounded arboricity, our algorithm finds a node of approximately maximal influence in sublinear time when $\beta = \omega(log^3(n))$. We note that many rich classes of graphs have bounded arboricity, including planar graphs, graphs with small maximum in-degree or maximum out-degree, other graphs with bounded treewidth (as the treewidth is at most twice the arboricity), many models of network formation and empirically observed social and technological networks [9, 18, 28].

We provide a lower bound of $\Omega((m+n)/\beta \cdot \min\{\beta, k\})$ on the runtime needed to obtain an $O(\beta)$-approximation. Thus, for networks with bounded arboricity, our algorithm is essentially runtime-optimal (up to logarithmic factors) given any fixed seed size k. Our method is randomized, and it succeeds with probability 3/5; moreover, we show that this probability can be amplified through repetition.

The intuition behind our modified algorithm is that a tradeoff between execution time and approximation factor can be achieved by constructing fewer edges in our hypergraph representation. Given an upper bound on runtime, we can build edges until that time has expired, then run the influence maximization algorithm using the resulting (impoverished) hypergraph. We show that this approach generates a solution whose quality degrades gracefully with the preprocessing time, with an important caveat. If there are many individuals with high influence, it may be that a reduction in runtime prevents us from achieving enough concentration to estimate the influence of any node. If so, the highest-degree node(s) in the constructed hypergraph will not necessarily be the nodes of highest influence in the original graph. However, in this case, the fact that many individuals have high influence enables an alternative approach: a node chosen at random, according to the degree distribution of nodes in the hypergraph representation, will have high influence with high probability.

Given the above, our algorithm will proceed by constructing two possible seed sets: one using the greedy algorithm applied to the constructed hypergraph, and the other is a singleton selected at random according to the hypergraph degree distribution. To decide between the two, we design a procedure for efficienctly estimating the influence of a given set, up to a maximum of n/β. We then return the set with higher tested influence.

Our test for estimating influence proceeds by repeatedly constructing depth-first trees of various sizes, rooted at the nodes to be tested. This is done in a careful way in order to keep the overall runtime small, which depends on the density of the densest subgraph in the network: if the nodes to be tested lie in a particularly dense region of the graph, extra time may be required to build partial spanning trees, even if the graph is sparse on average. This dependency is what motivates the arboricity term in the approximation factor of the algorithm.

Finally, we emphasize that our solution concept is to return a set with high expected influence, with respect to the influence diffusion process, with probability greater than 1/2 over randomness in the approximation algorithm. A potential relaxation would be to develop an algorithm that returns a set with high expected influence, where the expectation is with respect to both the diffusion process

[4]The arboricity of a network is the minimum number of spanning forests needed to cover all edges [35].

and the randomness in the approximation algorithm. For this weaker notion of approximability, the final testing phase of the algorithm described above is unnecessary: we could simply return one of our two potential solutions at random. This modified algorithm would have a runtime of $O(\frac{(n+m)\log^3(n)}{\beta})$, dropping the dependence on arboricity. As the lower bound of $\Omega((m+n)/\beta \cdot \min\{\beta, k\})$ holds even for this relaxed solution concept, this algorithm is nearly optimal (up to logarithmic factors) for arbitrary networks, given a fixed k. While we feel that the original (stronger) approximation notion is more relevant, this alternative formulation may be of interest in cases where variability in solution quality can be tolerated.

1.1 Related Work

Models of influence spread in networks, covering both cascade and threshold phenomena, are well-studied in the sociology and marketing literature [19, 22, 38]. The problem of finding the most influential set of nodes to target for a diffusive process was first posed by Domingos and Richardson [17, 37]. A formal development of the IC model, along with a greedy algorithm based upon submodular maximization, was given by Kempe et al. [24]. Many subsequent works have studied the nature of diffusion in online social networks, using empirical data to estimate influence probabilities and infer network topology; see [20, 30, 31].

It has been shown that many alternative formulations of the influence maximization problem are computationally difficult. The problem of finding, in a *linear threshold* model, a set of minimal size that influences the entire network was shown to be inapproximable within $O(n^{1-\epsilon})$ by Chen [12]. The problem of determining influence spread given a seed set in the IC model is #P-hard [13].

There has been a line of work aimed at improving the runtime of the algorithm by Kempe et al. [24]. These have focused largely on heuristics, such as assuming that all nodes have relatively low influence or that the input graph is clustered [13, 15, 26, 40], as well as empirically-motivated implementation improvements [14, 29]. One particular approach of note involves first attempting to sparsify the input graph, then estimating influence on the reduced network [15, 32]. Unfortunately, these sparsification problems are shown to be computationally intractible in general.

Various alternative formulations of influence spread as a submodular process have been proposed and analyzed in the literature [25, 34], including those that include interations between multiple diffusive processes [5, 21]. We focus specifically on the IC model, and leave open the question of whether our methods can be extended to apply to these alternative models.

The influence estimation problem shares some commonality with the problems of local graph partitioning, as well as estimating pagerank and personalized pagerank vectors [1, 2, 6, 39]. These problems admit local algorithms based on sampling short random walks. These methods do not seem directly applicable to influence maximization due to the inherently non-local nature of influence cascades.

2 Model and Preliminaries

The Independent Cascade Model In the independent cascade (IC) model, influence spreads via an edge-weighted directed graph $\mathcal{G}$. An infection begins at a set S of seed nodes, and spreads through the network in rounds. Each infected node v has a single chance, upon first becoming infected, of subsequently infecting his neighbors. Each directed edge $e = (v, u)$ has a weight $p_e \in [0, 1]$ representing the probability that the process spreads along edge e to node u in the round following the round in which v was first infected.

As noted in [24], the above process has the following equivalent description. We can interpret $\mathcal{G}$ as a distribution over unweighted directed graphs, where each edge e is independently realized with probability p_e. If we realize a graph G according to this probability distribution, then we can

associate the set of infected nodes in the original process with the set of nodes reachable from seed set S in G. We will make use of this alternative formulation of the IC model throughout the paper.

Notation We let m and n denote the number of edges and nodes, respectively, in the weighted directed graph $\mathcal{G}$. We write $G \sim \mathcal{G}$ to mean that G is drawn from the random graph distribution $\mathcal{G}$. Given set S of vertices and (unweighted) graph G, write $C_G(S)$ for the set of nodes reachable from S in G. When G is drawn from $\mathcal{G}$, we will refer to this as the set of nodes influenced by S. We write $I_G(S) = |C_G(S)|$ for the number of nodes influenced by S, which we call the influence of S in G. We write $\mathbb{E}_{\mathcal{G}}[I(S)] = \mathbb{E}_{G\sim\mathcal{G}}[I_G(S)]$ for the expected influence of S in $\mathcal{G}$.

Given two sets of nodes S and W, we write $C_G(S|W)$ for the set of nodes reachable from S but not from W. That is, $C_G(S|W) = C_G(S) \setminus C_G(W)$. As before, we write $I_G(S|W) = |C_G(S|W)|$; we refer to this as the marginal influence of S given W. The expected marginal influence of S given W is $\mathbb{E}_{\mathcal{G}}[I(S|W)] = \mathbb{E}_{G\sim\mathcal{G}}[I_G(S|W)]$.

In general, a vertex in the subscript of an expectation or probability denotes the vertex being selected uniformly at random from the set of vertices of $\mathcal{G}$. For example, $\mathbb{E}_{v,\mathcal{G}}[I(v)]$ is the average, over all graph nodes v, of the expected influence of v.

For a given graph G, define G^T to be the *transpose graph* of G: $(u, v) \in G$ iff $(v, u) \in G^T$. We apply this notation to both weighted and unweighted graphs.

The Influence Maximization Problem Given graph $\mathcal{G}$ and integer $k \geq 1$, the influence maximization problem is to find a set S of at most k nodes maximizing the value of $\mathbb{E}_{\mathcal{G}}[I(S)]$. For $\beta > 1$, we say that a particular set of nodes T with $|T| \leq k$ is a $\frac{1}{\beta}$-approximation to the influence maximization problem if $\mathbb{E}_{\mathcal{G}}[I(T)] \geq \frac{\max_{S:|S|=k} \mathbb{E}_{\mathcal{G}}[I(S)]}{\beta}$.

We assume that graph $\mathcal{G}$ is provided in adjacency list format, with the neighbors of a given vertex v ordered arbitrarily.

A Simulation Primitive Our algorithms we will make use of a primitive that realizes an instance of the nodes influenced by a given vertex u in weighted graph $\mathcal{G}$, and returns this set of nodes. Conceptually, this is done by realizing some $G \sim \mathcal{G}$ and traversing $C_G(u)$.

Let us briefly discuss the implementation of such a primitive. Given node u, we can run a depth first search in $\mathcal{G}$ starting at node u. Before traversing any given edge e, we perform a random test: with probability p_e we traverse the edge as normal, and with probability $1 - p_e$ we do not traverse edge e and ignore it from that point onward. The set of nodes traversed in this manner is equivalent to $C_G(u)$ for $G \sim \mathcal{G}$, due to deferred randomness. We then return the set of nodes traversed. The runtime of this procedure is precisely the sum of the degrees (in $\mathcal{G}$) of the vertices in $C_G(u)$.

We can implement this procedure for a traversal of G^T, rather than G, by following in-links rather than out-links in our tree traversal. Also, we will sometimes wish to run this procedure with an upper bound on the number of nodes to traverse; in this case we simply abort the depth-first traversal when the bound has been reached and return the set of nodes explored up to that point.

3 An Approximation Algorithm for Influence Maximization

In this section we present an algorithm for the influence maximization problem on arbitrary directed graphs. Our algorithm returns a $(1 - \frac{1}{e} - \epsilon)$-approximation to the influence maximization problem, with success probability 3/5, in time $O((m + n)\epsilon^{-3} \log n)$. We note that this algorithm is a simplification of a more general version that permits a tradeoff between runtime and approximation, which appears in Section 4.

Algorithm 1 Maximize Influence

Require: Precision parameter $\epsilon \in (0, 1)$, directed edge-weighted graph $\mathcal{G}$.

$R \leftarrow 72(m+n)\epsilon^{-3}\log(n)$
$\mathcal{H} \leftarrow$ BuildHypergraph(R)
return BuildSeedSet$(\mathcal{H}, k)$

BuildHypergraph(R)**:**

Initialize $\mathcal{H} = (V, \emptyset)$.
repeat
  Choose node u from $\mathcal{G}$ uniformly at random.
  Simulate influence spread, starting from u, in $\mathcal{G}^T$. Let T be the set of nodes discovered.
  Add T to the edge set of $\mathcal{H}$.
until R steps have been taken in total by the simulation process.
return $\mathcal{H}$

BuildSeedSet$(\mathcal{H}, k)$**:**

for $i = 1, \ldots, k$ **do**
  $v_i \leftarrow \text{argmax}_v\{deg_{\mathcal{H}}(v)\}$
  Remove v_i and all incident edges from $\mathcal{H}$
return $\{v_1, \ldots, v_k\}$

The algorithm is described formally as Algorithm 1, but let us begin by describing our construction informally. Our approach proceeds in two steps. The first step, BuildHypergraph, generates a sparse, randomized hypergraph representation $\mathcal{H}$ of our underlying graph G. This is done by repeatedly simulating the influence spread process on the transpose of the input graph, G^T. This simulation process is performed as described in Section 2: we begin at a random node u and proceed via depth-first search, where each encountered edge e is traversed independently with probability p_e. The set of nodes encountered becomes an edge in $\mathcal{H}$. We then repeat this process, generating multiple hyperedges. The BuildHypergraph subroutine takes as input a bound R on its runtime; we continue building edges until a total of R steps has been taken by the simulation process. (Note that the number of steps taken by the process is equal to the number of edges considered by the depth-first search process). Once R steps have been taken in total over all simulations, we return the resulting hypergraph.

In the second step, BuildSeedSet, we use our hypergraph representation to construct our output set. This is done by repeatedly choosing the node with highest degree in $\mathcal{H}$, then removing that node and all incident edges from $\mathcal{H}$. The resulting set of k nodes is the generated seed set.

While Algorithm 1 is relatively simple to describe, it is not obvious at all why such an algorithm should work well under its imposed, stringent time constraints. We now turn to provide a detailed analysis of Algorithm 1. Fix k and a weighted directed graph $\mathcal{G}$. Let $\text{OPT} = \max_{S:|S|=k}\{\mathbb{E}_{\mathcal{G}}[I(S)]\}$, the maximum expected influence of a set of k nodes.

Our goal is to bound the approximation factor of the set returned by Algorithm 1.

Theorem 3.1. *Fix $\epsilon > 0$. Algorithm 1 returns a set S with $\mathbb{E}_{\mathcal{G}}[I(S)] \geq (1 - \frac{1}{e} - \epsilon)\text{OPT}$, with probability at least $3/5$, and runs in time $O(\frac{(m+n)\log(n)}{\epsilon^3})$.*

The idea behind the proof of Theorem 3.1 is as follows. First, we observe that the influence of a set of nodes S is precisely n times the probability that a randomly selected node u influences any node from S in the transpose graph G^T.

Observation 3.2. *For each subset of nodes $S \subseteq \mathcal{G}$,*

$$\mathbb{E}_{G\sim\mathcal{G}}[I_G(S)] = n\mathrm{Pr}_{u,G\sim\mathcal{G}}[S \cap C_{G^T}(u) \neq \emptyset].$$

Proof.

$$\begin{aligned}
\mathbb{E}_{G\sim\mathcal{G}}[I_G(S)] &= \sum_{u\in G} \mathrm{Pr}_{G\sim\mathcal{G}}[\exists v \in S \text{ such that } u \in C_G(v)] \\
&= \sum_{u\in G} \mathrm{Pr}_{G\sim\mathcal{G}}[\exists v \in S \text{ such that } v \in C_{G^T}(u)] \\
&= n\mathrm{Pr}_{u,G\sim\mathcal{G}}[\exists v \in S \text{ such that } v \in C_{G^T}(u)] \\
&= n\mathrm{Pr}_{u,G\sim\mathcal{G}}[S \cap C_{G^T}(u) \neq \emptyset].
\end{aligned}$$

□

Observation 3.2 implies that we can estimate $\mathbb{E}_{\mathcal{G}}[I(S)]$ by estimating the probability of the event $S \cap C_{G^T}(u) \neq \emptyset$. The degree of a node v in $\mathcal{H}$ is precisely the number of times we observed that v was influenced by a randomly selected node u. We can therefore think of $\mathcal{H}$ as encoding an approximation to the influence function in graph $\mathcal{G}$.

We now show that the algorithm takes enough samples to accurately estimate the influences of the nodes in the network. This requires two steps. First, we show that runtime $R = 72(m+n)\epsilon^{-3}\log(n)$ is enough to build a sufficiently rich hypergraph structure, with high probability over the random outcomes of the influence cascade model.

Lemma 3.3. *Hypergraph $\mathcal{H}$ will contain at least $\frac{24n\log(n)}{\mathrm{OPT}\epsilon^3}$ edges, with probability at least $\frac{2}{3}$.*

Proof. Given a vertex u and an edge $e = (v, w)$, consider the random event indicating whether edge e is checked as part of the process of growing a depth-first search rooted at u in the IC process corresponding to graph $G^T \sim \mathcal{G}^T$. Note that edge e is checked if and only if node v is influenced by node u in this invocation of the IC process. In other words, edge $e = (v, w)$ is checked as part of the influence spread process on line 4 of BuildHypergraph if and only if $v \in T$. Write $m_{G^T}(u)$ for the random variable indicating the number of edges that are checked as part of building the influence set T starting at node u in G^T.

Let $X = \frac{24n\log(n)}{OPT\epsilon^3}$ for notational convenience. Consider the first (up to) X iterations of the loop on lines 2-6 of BuildHypergraph. Note that $\mathcal{H}$ will have at least X edges if the total runtime of the first X iterations is at most R. The expected runtime of the algorithm over these iterations is

$$\begin{aligned}
&X \cdot \mathbb{E}_{u,G\sim\mathcal{G}}[1 + m_{G^T}(u)] = X + \frac{X}{n}\mathbb{E}_{G\sim\mathcal{G}}\left[\sum_u m_{G^T}(u)\right] \\
&= X + \frac{24\log(n)}{OPT\epsilon^3}\mathbb{E}_{G\sim\mathcal{G}}\left[\sum_u m_{G^T}(u)\right] \\
&\leq X + \frac{24\log(n)}{OPT\epsilon^3} \sum_{e=(v,w)\in\mathcal{G}^T} \mathbb{E}_{G\sim\mathcal{G}}\left[|\{u : v \in C_{G^T}(u)\}|\right] \\
&= X + \frac{24\log(n)}{OPT\epsilon^3} \sum_{e=(v,w)\in\mathcal{G}^T} \mathbb{E}_{G\sim\mathcal{G}}\left[|\{u : u \in C_G(v)\}|\right] \\
&\leq \frac{24n\log(n)}{\epsilon^3} + \frac{24\log(n)}{OPT\epsilon^3} \sum_{e=(v,w)\in\mathcal{G}^T} OPT \\
&= \frac{24(m+n)\log(n)}{\epsilon^3}.
\end{aligned}$$

Here , the first inequality (line 4 from above) follows by noting that an edge $(v, w) \in \mathcal{G}^T$ is traversed as part of $m_{G^T}(u)$ if and only if v appears in $C_{G^T}(u)$.

Thus, by the Markov inequality, the probability that the runtime over the first X iterations is greater than $R = 72(m+n)\epsilon^{-3}\log(n)$ is at most $\frac{1}{3}$. The probability that at least X edges are present in hypergraph $\mathcal{H}$ is therefore at least $\frac{2}{3}$, as required. □

Next, we show that the resulting hypergraph is of sufficient size to estimate the influence of each set, up to an additive error that shrinks with ϵ, with probability $1 - 1/\text{POLY}(n)$. We then show that such estimation guarantees suffice to find good approximations to the influence maximization problem.

Write $m(\mathcal{H})$ and $deg_{\mathcal{H}}(S)$ for the number of edges of $\mathcal{H}$ and the number of edges from $\mathcal{H}$ incident with a node from S, respectively. An important subtlety is that the value of $m(\mathcal{H})$ is a random variable, determined by the stopping condition of BuildHypergraph. We must be careful to bound the effect on our influence estimation. We show that the value of $m(\mathcal{H})$ is sufficiently concentrated that the resulting bias is insignificant.

Lemma 3.4. *Suppose that $m(\mathcal{H}) \geq \frac{24n\log(n)}{\text{OPT}\epsilon^3}$. Then, for any set of nodes $S \subseteq V$, $\Pr[|\mathbb{E}_{\mathcal{G}}[I(S)] - \frac{n}{m(\mathcal{H})}deg_{\mathcal{H}}(S)| > \epsilon\text{OPT}] < \frac{1}{n^3}$, with probability taken over randomness in $\mathcal{H}$.*

Proof. By assumption we have that $m(\mathcal{H}) \geq 24(m+n)\log(n)/(\text{OPT}\epsilon^3)$, and we also know $m(\mathcal{H}) \leq R = 72(m+n)\log(n)/\epsilon^3$ (since each edge has size at least 1). Fix some arbitrary

$$T \in [\frac{24(m+n)\log(n)}{\text{OPT}\epsilon^3}, \frac{24(m+n)\log(n)}{\epsilon^3}].$$

We will study the probability that

$$|\mathbb{E}_{\mathcal{G}}[I(S)] - \frac{n}{m(\mathcal{H})}deg_{\mathcal{H}}(S)| > \epsilon\text{OPT}$$

conditional on $m(\mathcal{H}) = T$.

Suppose first that $\mathbb{E}_{\mathcal{G}}[I(S)] \geq \epsilon OPT$. Let D_S denote the degree of S in $\mathcal{H}$, for notational convenience. Thinking of D_S as a random variable, we have that D_S is the sum of $m(\mathcal{H}) = T$ identically distributed Bernoulli random variables each with probability $\mathbb{E}_{\mathcal{G}}[I(S)]/n \geq \epsilon OPT/n$, by Observation 3.2. In particular, since $m(\mathcal{H}) = T \geq 24(m+n)\log(n)/(OPT\epsilon^3)$,

$$\mathbb{E}[D_S] \geq \frac{\epsilon OPT}{n}m(\mathcal{H}) \geq 24\epsilon^{-2}\log n.$$

The Multiplicative Chernoff bound (A.1) then implies that

$$\begin{aligned}&\Pr\left[D_S < (1-\epsilon)\frac{m(\mathcal{H})}{n}\mathbb{E}_{\mathcal{G}}[I(S)]\right] = \Pr\left[D_S < (1-\epsilon)\mathbb{E}[D_S]\right] \\ &< e^{-\mathbb{E}[D_S]\epsilon^2/2} < e^{-12\log(n)} = \frac{1}{n^{12}}.\end{aligned}$$

Similarly, we can use the Multiplicative Chernoff bound to conclude that

$$\begin{aligned}&\Pr\left[D_S > (1+\epsilon)\frac{m(\mathcal{H})}{n}\mathbb{E}_{\mathcal{G}}[I(S)]\right] < e^{-\mathbb{E}[D_S]\epsilon^2/4} \\ &\leq e^{-6\log(n)} = \frac{1}{n^6}.\end{aligned}$$

Next suppose that $\mathbb{E}_{\mathcal{G}}[I(S)] < \epsilon OPT$, so that $\mathbb{E}[D_S] < \epsilon \cdot OPT \cdot m(\mathcal{H})/n$. In this case, the Multiplicative Chernoff bound implies that

$$\begin{aligned}
&\Pr\left[D_S > \mathbb{E}[D_S] + \frac{\epsilon OPT}{n} m(\mathcal{H})\right] \\
&= \Pr\left[D_S > \mathbb{E}[D_S]\left(1 + \frac{\epsilon OPT}{n}\frac{m(\mathcal{H})}{\mathbb{E}[D_S]}\right)\right] \\
&< e^{-\frac{\epsilon OPT}{n} m(\mathcal{H})/2} \\
&< e^{-12\log(n)} \leq \frac{1}{n^{12}}.
\end{aligned}$$

Thus, in all cases, the probability that the event of interest occurs is at most $\frac{1}{n^6}$. Since we conditioned on $m(\mathcal{H}) = T$, and since there are no more than $24(m+n)\log(n)/\epsilon^3 = o(n^3)$ potential values of T, the union bound implies that the unconditional probability that $|\mathbb{E}_{\mathcal{G}}[I(S)] - \frac{n}{m(\mathcal{H})} deg_{\mathcal{H}}(S)| > \epsilon \text{OPT}$ is at most $\frac{1}{n^3}$, as required. □

Finally, we must show that the greedy algorithm applied to $\mathcal{H}$ in BuildSeedSet returns a good approximation to the original optimization problem. Recall that, in general, the greedy algorithm for submodular function maximization proceeds by repeatedly selecting the singleton with maximal contribution to the function value, up to the cardinality constraint. The following lemma shows that if one submodular function is approximated sufficiently well by a distribution of submodular functions, then applying the greedy algorithm to a function drawn from the distribution yields a good approximation with respect to the original.

Lemma 3.5. *Choose $\delta > 0$ and suppose that $f\colon 2^V \to \mathbb{R}_{\geq 0}$ is a non-decreasing submodular function. Let D be a distribution over non-decreasing submodular functions with the property that, for all sets S with $|S| \leq k$, $\Pr_{\hat{f}\sim D}[|f(S) - \hat{f}(S)| > \delta] < 1/n^3$. If we write $S_{\hat{f}}$ for the set returned by the greedy algorithm on input $\hat{f}$, then*

$$\Pr_{\hat{f}\sim D}\left[f(S_{\hat{f}}) < (1-1/e)\left(\max_{S\colon |S|=k} f(S)\right) - 2\delta\right] < 1/n.$$

Proof. Choose $S^* \in \text{argmax}_{|S|=k}\{f(S)\}$. With probability at least $1 - 1/n^3$, $\hat{f}(S^*) \geq f(S^*) - \delta$. So, in particular, $\max_{|S|=k} \hat{f}(S) \geq f(S^*) - \delta$.

We run the greedy algorithm on function $\hat{f}$; let S_i be the set of nodes selected up to and including iteration i (with $S_0 = \emptyset$). On iteration i, we consider each set of the form $S_{i-1} \cup \{x\}$ where x is a singleton. There are at most n of these sets, and hence the union bound implies that f and $\hat{f}$ differ by at most δ on each of these sets, with probability at least $1 - 1/n^2$. In particular, $|f(S_i) - \hat{f}(S_i)| < \delta$. Taking the union bound over all iterations, we have that $|f(S_k) - \hat{f}(S_k)| < \delta$ with probability at least $1 - 1/n$. We therefore have

$$f(S_k) \geq \hat{f}(S_k) - \delta \geq (1-1/e)\max_S \hat{f}(S) - \delta \geq (1-1/e)f(S^*) - 2\delta$$

conditioning on an event of probability $1 - 1/n$. □

We are now ready to complete our proof of Theorem 3.1.

Proof of Theorem 3.1. Lemma 3.3 and 3.4 together imply that, conditioning on an event of probability at least 3/5, we will have

$$\Pr\left[\left|\mathbb{E}_{\mathcal{G}}[I(S)] - \frac{n \cdot deg_{\mathcal{H}}(S)}{m(\mathcal{H})}\right| > \epsilon \text{OPT}\right] < \frac{1}{n^3}$$

for each $S \subseteq V$. We then apply Lemma 3.5 with $f(S) := \mathbb{E}_{\mathcal{G}}[I(S)]$, $\hat{f}(S) := \frac{n \cdot deg_{\mathcal{H}}(S)}{m(\mathcal{H})}$ (drawn from distribution corresponding to distribution of $\mathcal{H}$ returned by BuildHypergraph), and $\delta = \epsilon \mathrm{OPT}$. Lemma 3.5 implies that, with probability at least $1 - \frac{1}{n}$, the greedy algorithm applied to $\mathcal{H}$ returns a set S with $\mathbb{E}_{\mathcal{G}[I(S)]} \geq (1 - 1/e)\mathrm{OPT} - 2\epsilon\mathrm{OPT} = (1 - 1/e - 2\epsilon)\mathrm{OPT}$. Noting that this is precisely the set returned by BuildSeedSet gives the desired bound on the approximation factor (rescaling ϵ by a factor of 2). Thus the claim holds with probability at least $2/3 - 1/n \geq 3/5$ (for $n \geq 20$).

Finally, we argue that our algorithm can be implemented in the appropriate runtime. The fact that BuildHypergraph executes in the required time follows from the explicit bound on its runtime. For BuildSeedSet, we will maintain a list of vertices sorted by their degree in $\mathcal{H}$; this will allow us to repeatedly select the maximum-degree node in constant time. The initial sort takes time $O(n \log n)$. We must bound the time needed to remove an edge from $\mathcal{H}$ and correspondingly update the sorted list. We will implement the sorted list as a doubly linked list of groups of vertices, where each group itself is implemented as a doubly linked list containing all vertices of a given degree (with only non-empty groups present). Each edge of $\mathcal{H}$ will maintain a list of pointers to its vertices. When an edge is removed, the degree of each vertex in the edge decreases by 1; we modify the list by shifting any decremented vertex to the preceding group (creating new groups and removing empty groups as necessary). Removing an edge from $\mathcal{H}$ and updating the sorted list therefore takes time proportional to the size of the edge. Since each edge in $\mathcal{H}$ can be removed at most once over all iterations of BuildSeedSet, the total runtime is at most the sum of node degrees in $\mathcal{H}$, which is at most $R = O((m + n)\epsilon^{-3} \log(n))$. □

3.1 Amplifying the Success Probability

Algorithm 1 returns a set of influence at least $(1 - \frac{1}{e} - \epsilon)$ with probability at least 3/5. The failure probability is due to Lemma 3.3: hypergraph $\mathcal{H}$ may not have sufficiently many edges after R steps have been taken by the simulation process in line 4 of the BuildHypergraph subprocedure. However, note that this failure condition is detectable via repetition: we can repeat Algorithm 1 multiple times, and use only the iteration that generates the most edges. The success rate can then be improved by repeated invocation, up to a maximum of $1 - 1/n$ with $\log(n)$ repetitions (at which point the error probability due to Lemma 3.4 becomes dominant).

We next note that, for any $\ell > 1$, the error bound in Lemma 3.4 can be improved to $\frac{1}{n^\ell}$, by increasing the value of R by a factor of ℓ, since this error derives from Chernoff bounds. This would allow the success rate of the algorithm to be improved up to a maximum of $1 - \frac{1}{n^\ell}$ by further repeated invocation. To summarize, the error rate of the algorithm can be improved to $1 - \frac{1}{n^\ell}$ for any ℓ, at the cost of increasing the runtime of the algorithm by a factor of $\ell^2 \log(n)$.

4 Approximate Influence Maximization in Sublinear Time

In this section we describe a modified algorithm that provides a tradeoff between runtime and approximation quality. For an an arbitrary $\beta > 1$, our algorithm will obtain a $\Theta(1/\beta)$-approximation to the influence maximization problem, in time $O(\frac{n \cdot a(\mathcal{G}) \log^3(n)}{\beta})$, where $a(\mathcal{G})$ is the arboricity of graph $\mathcal{G}$, with probability at least 3/5. The success rate can be improved by standard amplification techniques; we discuss this in Section 4.1.

Our algorithm is listed as Algorithm 2 below. The intuition behind our construction is as follows. We wish to find a set of nodes with high expected influence. One approach would be to apply Algorithm 1 and simply impose a tighter constraint on the amount of time that can be used constructing our hypergraph representation. This might correspond to reducing the value of parameter R by, say, a factor of β. Unfortunately, the precision of our sampling method does not always degrade gracefully with β: if β is sufficiently large, we may not have enough data to guess at

Algorithm 2 Influence Maximization with Tradeoff

Require: Approximation parameter $\beta > 1$, directed weighted graph $\mathcal{G}$.

$R \leftarrow \frac{(24 \cdot 36)(n+m)\log^2(n)}{\beta}$
$\mathcal{H} \leftarrow$ BuildHypergraph(R)
$S \leftarrow$ BuildSeedSet($\mathcal{H}, k$)
Choose $v \in V$ with probability proportional to degree in $\mathcal{H}$
if TestInfluence(S) > TestInfluence(v) **then return** S
else return $\{v\}$

TestInfluence(S)**:**
– τ: our guess at β times the influence of S.
– L: our guess at the realized influence size that contributes most to the expected influence.

for $\tau = n, n/2, n/4, \ldots, 1$ **do**
  for $L = n, n/2, n/4, \ldots, \tau$ **do**
    for $j = 1, \ldots, 32(L/\tau)\log(n)$ **do**
      Simulate influence in $\mathcal{G}$, starting from S, to a maximum of L/β distinct nodes.
      Let T_j be the set of nodes discovered.
      if $\sum_j |T_j| > 96(n/\beta)\log(n)$ **then return** τ/β
    if at least $256\log(n)$ sets T_j satisfy $|T_j| \geq L/\beta$ **then return** τ/β
return 1

a maximum-influence node (even if we allow ourselves a factor of β in the approximation ratio). In these cases, the sampling approach fails to provide a good approximation.

However, as we will show, our sampling fails precisely because many of the edges in our hypergraph construction were large, and (with constant probability) this can occur only if many of the nodes that make up those edges have high influence. In this case, we could proceed by selecting a node from the hypergraph at random, with probability proportional to its hypergraph degree. We prove that this procedure is likely to return a node of very high influence precisely in settings where the original approach is not.

However, for this to work we need to figure out which of the two approaches will succeed (since, in particular, the required bound on the hypergraph size is a function of OPT, which is unknown). We therefore apply both methods, then directly estimate the influence of each solution. We must do so carefully in order to keep our runtime small. To this end, the TestInfluence procedure generates a rough sketch of the probability distribution over influence values by repeatedly growing depth-first trees of various sizes. By balancing tree sizes with sampling precision, we are able to give an estimate of influence up to a maximum of n/β. We then return whichever solution has the greater estimated influence.

We note that, since our final estimation step proceeds by repeatedly exploring subgraphs of an input graph $\mathcal{G}$, its runtime will be tied to the *arboricity* of $\mathcal{G}$. Indeed, the arboricity is used to explicitly bound the time needed to construct a depth-first tree of a given size.

Definition 1. *The arboricity number of an undirected graph $\mathcal{G}$, denoted as $a(\mathcal{G})$, is the minimum number of spanning forests needed to cover all the edges of the graph. The arboricity of a directed, weighted graph is the arboricity of the undirected, unweighted version of the graph.*

Theorem 4.1. *For any $\beta > 1$, Algorithm 2 returns, with probability of at least $3/5$, a node with expected influence at least $\min\{\frac{1}{2}, \frac{1}{\beta}\} \cdot OPT$. Its runtime is $O(\frac{n \cdot a(\mathcal{G})\log^3(n)}{\beta})$.*

Before proving Theorem 4.1, let us discuss its statement and some minor variations.

Discussion: It will turn out that the runtime of Algorithm 2 is dominated by subprocedure TestInfluence: the time needed to estimate the influence of a given solution. Given access to an oracle that estimates the influence of a given set, up to a maximum of n/β, the runtime of Algorithm 2 becomes $O(\frac{(n+m)\log^3(n)}{\beta} + X(n,m,\beta))$, where $X(n,m,\beta)$ is a bound on the runtime of the oracle. We provide an implementation of such an oracle, based on growing spanning forests. Our analysis is given in terms of the arboricity of the underlying graph and shows that $X(n,m,\beta) \subseteq O(\frac{n \cdot a(\mathcal{G})\log^3(n)}{\beta})$. One potential method for improving our analysis would be to find a more efficient impliementation of an influence-approximating oracle.

Another important note is in regards to the probabilistic guarantee we require from our algorithm. Algorithm 2 returns a set that has high influence (in expectation over the random diffusion process), with probability at least 3/5 over the random bits of the algorithm. The runtime of Algorithm 2 can be improved if our goal is relaxed to returning a set of high influence in expectation both over the random diffusion process and the randomness in the algorithm. For this relaxed solution concept, it becomes unnecessary to test the influence of the two potential solutions: the algorithm can simply choose between them at random. The expected influence of the set returned would then be at least $\text{OPT}/2\beta$. The runtime of this modified algorithm is $O(\frac{(n+m)\log^3(n)}{\beta})$. □

Let us now turn to the proof of Theorem 4.1. Observe that when $\beta = O(\log n)$, the conditions of Theorem 4.1 are satisfied by Algorithm 1 with (say) $\epsilon = 1/6$ (to get an approximation factor bigger than 1/2), since its runtime is $O((m+n)\log(n)) \subseteq O(\frac{n \cdot a(\mathcal{G})\log^3(n)}{\beta})$ (see Fact 4.5 in the appendix). We can therefore assume that $\beta > \log(n)$.

Our analysis proceeds via two cases, depending on whether $\mathcal{H}$ has sufficiently many edges as a function of OPT. We first show that, subject to $\mathcal{H}$ having many edges, set S from line 3 is likely to have high influence. This follows the analysis from Theorem 3.1 almost exactly.

Lemma 4.2. *Suppose that $m(\mathcal{H}) \geq \frac{24n\log(n)}{OPT}$. Then, with probability at least $1 - \frac{1}{n}$, set S satisfies $\mathbb{E}_{\mathcal{G}}[I(S)] \geq \frac{1}{2}OPT$, with probability taken over randomness in $\mathcal{H}$.*

Proof. This follows by applying Lemma 3.4 with $\epsilon = \frac{1}{6}$, followed by the analysis of BuildSeedSet$(\mathcal{H}, k)$ from the proof of Theorem 3.1. □

Note that β does not appear explicitly in the statement of Lemma 4.2. The (implicit) role of β in Lemma 4.2 is that as β becomes large, Algorithm 2 uses fewer steps to construct hypergraph $\mathcal{H}$ and hence the condition of the lemma is less likely to be satisfied.

We next show that if $m(\mathcal{H})$ is small, then node v from line 4 is likely to have high influence. Recall our assumption that $\beta > \log(n)$.

Lemma 4.3. *Suppose that $m(\mathcal{H}) < \frac{24n\log(n)}{OPT}$. Then, with probability at least 2/3, node v satisfies $\mathbb{E}_{\mathcal{G}}[I(v)] \geq \text{OPT}\log(n)/\beta$, with probability taken over randomness in $\mathcal{H}$.*

Proof. Let random variable X denote the number of times that a node with influence at most $\text{OPT}\log(n)/\beta$ was added to a hyperedge of $\mathcal{H}$. Since $\mathcal{H}$ has fewer than $\frac{24n\log(n)}{\text{OPT}}$ edges, the expected value of X is at most

$$\begin{aligned} E[X] &\leq \frac{24n\log(n)}{\text{OPT}} \sum_{u \in V} \frac{1}{n} \min\{\mathbb{E}_{\mathcal{G}}[I(u)], \text{OPT}\log(n)/\beta\} \\ &\leq \frac{24n\log^2(n)}{\beta}. \end{aligned}$$

Markov inequality then gives that $\Pr[X > \frac{(24\cdot 6)n\log^2(n)}{\beta}] < 1/6$. Conditioning on this event, we have that at most $\frac{(24\cdot 6)n\log^2(n)}{\beta}$ of the nodes touched by BuildHypergraph have influence less than

$OPT \log(n)/\beta$. Since at least $\frac{(24 \cdot 36) n \log^2(n)}{\beta}$ nodes were touched in total, the probability that node v from line 4 has influence less than $OPT \log(n)/\beta$ is at most 1/6. The union bound then allows us to conclude that v has $\mathbb{E}[I(v)] \geq OPT/\beta$ with probability at least $1 - (1/6 + 1/6) \geq 2/3$. □

Next, we show that procedure TestInfluence(S) returns a sufficiently good estimate of the influence of a given set of nodes. The proof, which is somewhat technical, is deferred to Section 4.3.

Lemma 4.4. *Given a set of nodes S procedure TestInfluence returns a value from the range $[x/\log(n), x]$ with probability at least $1 - \frac{1}{2n}$, where $x = \min\{\mathbb{E}_{\mathcal{G}}[I(S)], n/\beta\}$.*

Finally, we show that the runtime of Algorithm 2 is at most $O(\frac{n \cdot a(\mathcal{G}) \log^3(n)}{\beta})$. The following fact about arboricity will be particularly useful.

Fact 4.5 (Nash-Williams [35])**.** *Given any graph G and subgraph H of G (containing at least two nodes), $\frac{m(H)}{n(H)-1} \leq a(G)$.*

As noted by Nash-Williams this characterization of arboricity is tight as there is always one such H with $\lceil \frac{m(H)}{n(H)-1} \rceil = a(G)$. We are now ready to bound the runtime of Algorithm 2.

Lemma 4.6. *Algorithm 2 runs in time $O(\frac{n \cdot a(\mathcal{G}) \log^3(n)}{\beta})$.*

Proof. Similarly to the analysis of Algorithm 1, lines 1-3 take at most $R = O(\frac{(n+m) \log^2(n)}{\beta})$. Fact 4.5 implies that this is $O(\frac{n \log^2(n) \cdot a(\mathcal{G})}{\beta})$.

We must now show that calling TestInfluence() costs at most $O(\frac{n \log^3(n) \cdot a(\mathcal{G})}{\beta})$ which completes the proof. On each choice of L and τ TestInfluence() grows several trees, where the sum over those tree sizes is at most $O(\frac{n}{\beta}) \log n$. By the Nash-Williams theorem [35], each subgraph explored during this process has arboricity at most $a(\mathcal{G})$.

Thus the total time to build the trees (for a given τ and L) is most $O(\frac{n \cdot a(\mathcal{G})}{\beta} \log(n))$. To implement the test on line 7, for each value of L, one needs to keep track on the number of trees with size at least $\frac{L}{\beta}$. This can easily be done online by keeping a counter, updating the counter after each tree is realized. Since we also iterate over $\log(n)$ values for L and τ, the total runtime is $O(\frac{n \log^3(n) \cdot a(\mathcal{G})}{\beta})$. □

We are now ready to complete the proof of Theorem 4.1.

Proof of Theorem 4.1. Lemma 4.2 and Lemma 4.3 imply that, with probability at least $2/3 - 1/n^2 \geq 3/5$ (for $n \geq 5$), one of S or $\{v\}$ has influence at least $OPT \log(n)/\beta$ (recalling that $\beta > \log n$). Since TestInfluence determines the influence of each set up to a potential under-valuation of a factor of $\log(n)$, the set for which TestInfluence returns the highest estimate must therefore have influence at least $\frac{\mathrm{OPT} \log(n)}{\beta} \cdot \frac{1}{\log(n)} = \mathrm{OPT}/\beta$. The required bound on the runtime of Algorithm 2 follows directly from Lemma 4.6. □

4.1 Amplifying the Success Probability

Algorithm 2 returns, with probability 3/5, a set that approximates the maximum expected influence of a node in $\mathcal{G}$. We note that this returned set comes with an estimate of the optimal influence in the network; the "failure" conditions correspond to selecting this estimate incorrectly. To amplify success probability, we could return this estimate along with the set S. One could then call the algorithm multiple times; each successful invocation would generate an estimate at least OPT/β, whereas failed invocations would potentially generate smaller estimates. Amplification of success

probability then corresponds to accepting whichever output is associated with the highest estimate. This would allow success probability to be raised up to $1 - 1/n$, at which point sampling error in Lemma 4.2 becomes the dominant error factor.

As in our original algorithm, the error bound in Lemma 4.2 can be improved to $1 - \frac{1}{n^\ell}$ by increasing the value of R by a factor of ℓ, since this error derives from Chernoff bounds. This would allow the success rate of the algorithm to be improved up to a maximum of $1-\frac{1}{n^\ell}$ by further repeated invocation. To summarize, the error rate of the algorithm can be improved to $1 - \frac{1}{n^\ell}$ for any ℓ, at the cost of increasing the runtime of the algorithm by a factor of $\ell^2 \log(n)$.

4.2 A Lower Bound

We now provide a lower bound on the time it takes to compute a β-approximation for the maximum expected influence problem. In particular, for any given budget k, at least $\Omega(n/\beta)$ queries are required to obtain approximation factor $\frac{1}{\beta}$ with constant probability.

Theorem 4.7. *Let $0 < \epsilon < 1$, $\beta > 1$ be given. Any (possibly randomized) algorithm for the maximum influence problem that has runtime of $\frac{m+n}{24\beta \min\{k,\beta\}}$ cannot return, with probability at least $1 - \frac{1}{e} - \epsilon$, a set of nodes with approximation ratio better than $\frac{1}{\beta}$.*

Proof. Note first that for a graph consisting of n singletons, an algorithm must return at least k/β nodes to obtain an approximation ratio of $\frac{1}{\beta}$. Doing so in at most $n/2\beta^2$ queries requires that $2k/\beta \leq n/\beta^2$, which implies $2k\beta \leq n$. We can therefore assume $2k\beta \leq n$ for the remainder of the proof.

Consider the behavior of a randomized algorithm A. Assume for notational simplicity that β is an integer. We will build a family of lower bound graphs, one for each value of n (beginning from $n = \beta+1$); each graph will have $m \leq n$, so it will suffice to demonstrate a lower bound of $\frac{n}{12\beta \min\{k,\beta\}}$.

For a given value β the graph would be made from k components of size 2β and $n-2k\beta$ singleton components (recall that $2k\beta \leq n$). If algorithm A returns nodes from ℓ of the k components of size 2β, it achieves a total influence of $2\ell\beta + (k - \ell)$. Thus, to attain approximation factor better than $\frac{1}{\beta}$, we must have $2\ell\beta + (k - \ell) \geq \frac{1}{\beta}2k\beta$, which implies $\ell \geq \frac{k}{2\beta-1}$ for any $\beta > 1$.

Suppose $k > 12\beta$. The condition $\ell \geq \frac{k}{2\beta-1}$ implies that at least $\frac{k}{2\beta-1}$ of the large components must be queried by the algorithm, where each random query has probability $\frac{2k\beta}{n}$ of hitting a large component. If the algorithm makes fewer than $\frac{n}{12\beta^2}$ queries, then the expected number of components hit is $\frac{n}{12\beta^2} \cdot \frac{2k\beta}{n} = \frac{k}{6\beta}$. Multiplicative chernoff bounds then imply that the probability hitting more than $\frac{k}{2\beta}$ components is no more than $1 - e^{-\frac{k}{6\beta} \cdot 4/4} > 1 - 1/e$, a contradiction.

If $k \leq 12\beta$ then we need that $\ell \geq 1$, which occurs only if the algorithm queries at least one of the $k\beta$ vertices in the large components. With $\frac{n}{2k\beta}$ queries, for n large enough, this happens with probability smaller than $\frac{1}{e} + \epsilon$, a contradiction.

We conclude that, in all cases, at least $\frac{n}{12\beta \min\{k,\beta\}}$ queries are necessary to obtain approximation factor better than $\frac{1}{\beta}$ with probability at least $1 - \frac{1}{e} - \epsilon$, as required.

Finally, we note that our construction can be modified to apply to non-sparse networks, as follows. For any $d \leq n$, we can augment our graph by overlaying a d-regular graph with exponentially small weight on each edge. This does not significantly impact the influence of any set of nodes, but it increases the time to determine whether a node is in a large component by a factor of $O(d)$ (as edges must be traversed until one with non-exponentially-small weight is found). Thus, for each $d \leq n$, we have a lower bound of $\frac{nd}{24\beta \min\{k,\beta\}}$ for networks with $m = nd$. □

Discussion: The lower bound construction of Theorem 4.7 is tailored to the query model considered in this paper. In particular, we assume that vertices are not sorted by degree, component size, etc.

However, the construction can be easily modified to be robust to various changes in the model, by (for example) adding edges with small weight so that the exhibited network $\mathcal{G}$ becomes connected and/or d-regular for some fixed d (independent of n). □

4.3 Proof of Lemma 4.4

First recall the statement of Lemma 4.4. We must show that, given a set of nodes S, with probability at least $1 - \frac{1}{2n}$, procedure TestInfluence returns a value from the range $[x/\log(n), x]$, where $x = \min\{\mathbb{E}_{\mathcal{G}}[I(S)], n/\beta\}$. The key to algorithm TestInfluence is to efficiently decide, for a given set S and potential influence value $\tau \leq n/\beta$, whether S has influence at least $\tau \log(n)$ or less than τ. We begin with two useful observations that will assist us in this task.

Lemma 4.8. *For any $v \in \mathcal{G}$, there exists some t, $1 \leq t \leq \log n$, such that $2^t \Pr_{G\sim\mathcal{G}}[I_G(v) \geq 2^t] \geq \mathbb{E}_{G\sim\mathcal{G}}[I_G(v)]/\log(n)$.*

Proof.

$$\begin{aligned}\mathbb{E}_G[I(v)] &= \sum_{y=1}^{n} \Pr_{G\sim\mathcal{G}}[I_G(v) \geq y] \\ &= \sum_{t=0}^{\log n} \sum_{y=2^t}^{2^{t+1}} \Pr_{G\sim\mathcal{G}}[I_G(v) \geq y] \\ &\leq \sum_{t=0}^{\log n} 2^t \Pr_{G\sim\mathcal{G}}[I_G(v) \geq 2^t]\end{aligned}$$

and hence $\mathbb{E}_{G\sim\mathcal{G}}[I_G(v)] \leq \log(n) \max_t\{2^t \Pr_{G\sim\mathcal{G}}[I_S(v) \geq 2^t]\}$, as required. □

Lemma 4.9. *For any $v \in \mathcal{G}$ and any $1 \leq t \leq \log n$, $2^t \Pr_{G\sim\mathcal{G}}[I_G(v) \geq 2^t] \leq \mathbb{E}_{G\sim\mathcal{G}}[I_G(v)]$.*

Proof.

$$\begin{aligned}\mathbb{E}_{G\sim\mathcal{G}}[I_G(v)] &= \sum_{y=1}^{n} y \Pr_{G\sim\mathcal{G}}[I_G(v) = y] \\ &\leq \sum_{y=2^t}^{n} 2^t \Pr_{G\sim\mathcal{G}}[I_G(v) = y] \\ &= 2^t \Pr_{G\sim\mathcal{G}}[I_G(v) \geq 2^t].\end{aligned}$$

□

Note that we think of L in algorithm TestInfluence(S) as a guess at the value of 2^t from the statement of Lemma 4.8. We are now ready to proceed with the proof of Lemma 4.4.

Consider a given iteration of TestInfluence, corresponding to a value of τ. We think of τ as a guess at the value of $\mathbb{E}_{\mathcal{G}}[I(S)]\beta$. We will first show that, with high probability, we do not return on an iteration in which $\mathbb{E}_{G\sim\mathcal{G}}[I(S)] < \tau/\beta$. We must consider the return conditions on line 6 and line 7.

Suppose $\mathbb{E}_{G\sim\mathcal{G}}[I_G(S)] < \tau/\beta$. Consider the return condition on line 7. For set S, $\Pr_{G\sim\mathcal{G}}[I_G(S) \geq L/\beta] < (\tau/L)$ for each choice of L, by Lemma 4.9. Thus, for any given choice of L, the probability that the event $[I_G(S) \geq L/\beta]$ occurs more than $256\log(n)$ times in $(L/\tau)32\log(n)$ trials is (by multiplicative Chernoff bounds) at most $e^{-4\log(n)} = 1/n^4$. Taking the union bound over all values

of L, we conclude that with probability at least $1 - 1/n^2$ we will not return on an iteration for which $\mathbb{E}_{G\sim\mathcal{G}}[I_G(S)] < \tau/\beta$ on line 7.

Next consider the condition on line 6. Suppose $\mathbb{E}_{G\sim\mathcal{G}}[I_G(S)] < \tau/\beta$, and pick some $L' \in \{n, n/2, \ldots, \tau\}$. As above, $\Pr_{G\sim\mathcal{G}}[I_G(S) \geq L'/\beta] < (\tau/L')$. Consider the sets of nodes built for node S on the iteration corresponding to some choice of L. If $L \leq L'$, then each set T_j will have size at most L'/β. For any $L > L'$, we know that by the multiplicative chernoff bound, the probability that event $[I_G(S) \geq L'/\beta]$ occurs more than $2(L/\tau)32\log(n) \cdot (\tau/L') = 64(L/L')\log(n)$ times is at most $e^{-4\log(n)} = 1/n^4$. Taking the union bound over all choices of L and L', we have that with probability at least $1 - 1/n^2$, for each L and $L' \leq L$, fewer than $2(L/L')32\log(n)$ trees built on iteration L have size greater than L'/β. Thus, conditioning on this event, we have that the sum of tree sizes is less than

$$\sum_{t=1}^{\log L} (2^t/\beta)2(L/2^t)\log^2(n) \leq 2\frac{n}{\beta}32\log^3(n).$$

Taking the union bound over all values of L, we conclude that with probability at least $1 - 1/n^2$ we will not return on an iteration for which $\mathbb{E}_{G\sim\mathcal{G}}[I_G(S)] < \frac{\tau}{\beta}$, on line 6.

We will now show that, in an iteration in which $\tau\log(n)/\beta \leq \mathbb{E}_{\mathcal{G}}[I(S)]$, the algorithm returns with high probability. Suppose S has $\mathbb{E}_{G\sim\mathcal{G}}[I_G(S)] \geq (\tau/\beta)\log(n)$. Then, by Lemma 4.8, there exists some $L \geq \tau$, L a power of 2, such that $\Pr_{G\sim\mathcal{G}}[I_G(S) \geq L/\beta] \geq \frac{\tau}{2L}\log(n)$. Multiplicative chernoff bounds imply that, during the $32(L/\tau)\log(n)$ trees built for set S, the probability that fewer than $\frac{1}{2}32\log(n) = 16\log(n)$ reach size L/β is at most $e^{-4\log(n)} = 1/n^4$. Thus, assuming our algorithm does not return on line 6, it will return on line 7 with probability at least $1 - O(1/n^4)$, as required.

5 Acknowledgments

We thank Elchanan Mossel for helpful discussions.

References

[1] R. Andersen, C. Borgs, J. Chayes, J. Hopcraft, V. S. Mirrokni, and S.-H. Teng. Local computation of pagerank contributions. In *WAW*, pages 150–165, Berlin, Heidelberg, 2007. Springer-Verlag.

[2] R. Andersen, F. Chung, and K. Lang. Local graph partitioning using pagerank vectors. In *FOCS*, pages 475–486, 2006.

[3] E. Bakshy, B. Karrer, and L. A. Adamic. Social influence and the diffusion of user-created content. In *ACM EC*, pages 325–334, 2009.

[4] O. Ben-Zwi, D. Hermelin, D. Lokshtanov, and I. Newman. Treewidth governs the complexity of target set selection. *Disc. Opt.*, 8(1):87–96, 2011.

[5] S. Bharathi, D. Kempe, and M. Salek. Competitive influence maximization in social networks. In *WINE*, pages 306–311, 2007.

[6] C. Borgs, M. Brautbar, J. T. Chayes, and S.-H. Teng. A sublinear time algorithm for pagerank computations. In *WAW*, pages 41–53, 2012.

[7] M. Brautbar and M. Kearns. Local algorithms for finding interesting individuals in large networks. In *ICS*, pages 188–199, 2010.

[8] J. J. Brown and P. H. Reingen. Social ties and word of mouth referral behavior. *Journal of Consumer Research*, 14(3):350–362, 1987.

[9] S. Carmi, S. Havlin, S. Kirkpatrick, Y. Shavitt, and E. Shir. A model of internet topology using k-shell decomposition. *PNAS*, pages 11150–11154, 2007.

[10] D. Centola and M. Macy. Complex contagions and the weakness of long ties. *American Journal of Sociology*, 113(3):702–734, 2007.

[11] M. Cha, A. Mislove, and P. K. Gummadi. A measurement-driven analysis of information propagation in the flickr social network. In *WWW*, pages 721–730, 2009.

[12] N. Chen. On the approximability of influence in social networks. In *SODA*, pages 1029–1037, 2008.

[13] W. Chen, C. Wang, and Y. Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *KDD*, pages 1029–1038, 2010.

[14] W. Chen, Y. Wang, and S. Yang. Efficient influence maximization in social networks. In *KDD*, pages 199–208, 2009.

[15] W. Chen, Y. Yuan, and L. Zhang. Scalable influence maximization in social networks under the linear threshold model. In *ICDM*, pages 88–97, 2010.

[16] P. Dodds and D. Watts. Universal behavior in a generalized model of contagion. *Phys Rev Lett*, 92(21):218701, 2007.

[17] P. Domingos and M. Richardson. Mining the network value of customers. In *KDD*, pages 57–66, 2001.

[18] G. Goel and J. Gustedt. Bounded arboricity to determine the local structure of sparse graphs. In *WG*, pages 159–167, 2006.

[19] J. Goldenberg, B. Libai, and E. Mulle. Talk of the network: A complex systems look at the underlying process of word-of-mouth. *Marketing Letters*, pages 221–223, 2001.

[20] M. Gomez-Rodriguez, J. Leskovec, and A. Krause. Inferring networks of diffusion and influence. *TKDD*, 5(4):21, 2012.

[21] S. Goyal and M. Kearns. Competitive contagion in networks. In *STOC*, pages 759–774, 2012.

[22] M. Granovetter. Threshold models of collective behavior. *American Journal of Sociology*, (83):1420–1443, 1978.

[23] Q. Jiang, G. Song, G. Cong, Y. Wang, W. Si, and K. Xie. Simulated annealing based influence maximization in social networks. In *AAAI*, 2011.

[24] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In *KDD*, pages 137–146, 2003.

[25] D. Kempe, J. M. Kleinberg, and É. Tardos. Influential nodes in a diffusion model for social networks. In *ICALP*, pages 1127–1138, 2005.

[26] M. Kimura and K. Saito. Tractable models for information diffusion in social networks. In *PKDD*, pages 259–271, 2006.

[27] J. Leskovec, L. A. Adamic, and B. A. Huberman. The dynamics of viral marketing. *TWEB*, 1(1), 2007.

[28] J. Leskovec and E. Horvitz. Planetary-scale views on a large instant-messaging network. In *WWW*, pages 915–924, 2008.

[29] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. M. VanBriesen, and N. S. Glance. Cost-effective outbreak detection in networks. In *KDD*, pages 420–429, 2007.

[30] J. Leskovec, M. McGlohon, C. Faloutsos, N. S. Glance, and M. Hurst. Patterns of cascading behavior in large blog graphs. In *SDM*, 2007.

[31] D. Liben-Nowell and J. Kleinberg. Tracing information flow on a global scale using internet chain-letter data. *PNAS*, 105(12):4633–4638, 2008.

[32] M. Mathioudakis, F. Bonchi, C. Castillo, A. Gionis, and A. Ukkonen. Sparsification of influence networks. In *KDD*, pages 529–537, 2011.

[33] S. Morris. Contagion. *Review of Economic Studies*, 67:57–78, 2000.

[34] E. Mossel and S. Roch. On the submodularity of influence in social networks. In *STOC*, pages 128–134, 2007.

[35] C. S. A. Nash-Williams. Decomposition of finite graphs into forests. *Journal of the London Mathematical Society*, 39(1):12, 1964.

[36] D. Peleg. Local majorities, coalitions and monopolies in graphs: a review. *Theor. Comput. Sci.*, 282(2):231–257, 2002.

[37] M. Richardson and P. Domingos. Mining knowledge-sharing sites for viral marketing. In *KDD*, pages 61–70, 2002.

[38] E. Rogers. *Diffusion of Innovations*. Free Press, 5th edition, 2003.

[39] D. A. Spielman and S.-H. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *STOC*, pages 81–90, 2004.

[40] Y. Wang, G. Cong, G. Song, and K. Xie. Community-based greedy algorithm for mining top-k influential nodes in mobile social networks. In *KDD*, pages 1039–1048, 2010.

A Concentration Bounds

Lemma A.1. *(Multiplicative Chernoff Bound) Let X_i be n i.i.d. Bernoulli random variables with expectation μ each. Define $X = \sum_{i=1}^{n} X_i$. Then,*
For $0 < \lambda < 1$, $Pr[X < (1-\lambda)\mu n] < \exp(-\mu n \lambda^2/2)$.
For $0 < \lambda < 1$, $Pr[X > (1+\lambda)\mu n] < \exp(-\mu n \lambda^2/4)$.
For $\lambda > 1$, $Pr[X > (1+\lambda)\mu n] < \exp(-\mu n \lambda/2)$.

Lemma A.2. *(Additive Chernoff Bound) Let X_i be n i.i.d. Bernoulli random variables with expectation μ each. Define $X = \sum_{i=1}^{n} X_i$. Then, for $\lambda > 0$,*
$Pr[X < \mu n - \lambda] < \exp(-2\lambda^2/n)$.
$Pr[X > \mu n + \lambda] < \exp(-2\lambda^2/n)$.