Maximizing Social Influence in Nearly Optimal Time

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

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. Runtime is a primary consideration for this problem due to the massive size of the relevant input networks.

We provide a fast algorithm for the influence maximization problem, obtaining the near-optimal approximation factor of $(1 - \frac{1}{e} - \epsilon)$, for any $\epsilon > 0$, in time $O((m + n)\epsilon^{-3} \log n)$. Our algorithm is runtime-optimal (up to a logarithmic factor) and substantially improves upon the previously best-known algorithms which run in time $\Omega(mnk \cdot \text{POLY}(\epsilon^{-1}))$. Furthermore, our algorithm can be modified to allow early termination: if it is terminated after $O(\beta(m+n) \log n)$ steps for some $\beta < 1$ (which can depend on n), then it returns a solution with approximation factor $O(\beta)$. Finally, we show that this runtime is optimal (up to logarithmic factors) for any β and fixed seed size k.

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 [35, 8, 17, 9, 3, 10, 19]. 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 [16, 23, 26, 17].

This application gives rise to an algorithmic prob-

lem: given a network, how can we determine which individuals should be targeted to maximize the magnitude of a resulting cascade [16, 34, 23]? 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. This problem has been studied for various models of influence spread, leading to the development of polynomial-time algorithms that achieve constant approximations [23, 24, 32].

Relevant networks for this problem can have massive size, on the order of billions of edges. The running time of an influence maximization algorithm is therefore a primary consideration. This is compounded by the fact that social network data and influence parameters tend to be volatile, necessitating recomputation of solutions over time. For these reasons, near-linear runtime is a practical necessity for algorithms that work with massive network data. This stringent runtime requirement has spawned a large body of work aimed at developing fast, heuristic methods of finding influential individuals in social networks, despite the existence of the above-mentioned approximation algorithms. See, for example, [13, 12, 14, 27, 38, 30, 22, 25]. However, to date, this line of work has focused primarily on empirical methods. Currently, the fastest algorithms with constant-factor approximation guarantees have runtime $\Omega(nmk)$ [13].

In this paper we bridge this gap by developing a constant-factor approximation algorithm for the influence maximization problem, under the standard *independent cascade* model of influence spread, that runs in quasilinear time. Our algorithm can also be modified to run in sublinear time, with a correspondingly reduced approximation factor. Before describing these results in detail, we first provide some background into the influence model.

The Model: Independent Cascades We adopt the *independent cascade* (IC) model of diffusion, formalized by Kempe et al. [23]. 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

---
*Microsoft Research
†Computer and Information Science, University of Pennsylvania. Now in The Laboratory for Information and Decision Systems, Massachusetts Institute of Technology.
‡Microsoft Research
§Microsoft Research

$e = (v, u)$ 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 $|S| \leq k$.

The IC model captures the intuition that influence can spread stochastically through a network, much like a disease [23, 17, 15]. Since its introduction, it has become one of the prominent models of influence spread; see for example [13, 12, 14, 27, 38, 24]. Kempe et al. show that $\mathbb{E}[I(\cdot)]$ is a submodular monotone function [23], 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 [33, 31, 11, 4].

The greedy approach to maximizing influence in the IC model described above takes time $O(kn)$, *given oracle access to the function* $\mathbb{E}[I(\cdot)]$. However, influence values must be computed from the underlying network topology, by (for example) repeated simulation of the diffusion process. This leads ultimately[1] to a runtime[2] of $\Omega(mnk \cdot \text{POLY}(\epsilon^{-1}))$.

Our Result: A Quasi-Linear Time Algorithm Our main result is an algorithm for finding $(1 - 1/e - \epsilon)$-approximately optimal seed sets in arbitrary directed networks, which runs in time $O((m + n)\epsilon^{-3} \log n)$. Notably, the runtime of our algorithm is independent of the number of seeds k. Moreover, this runtime is close to optimal, as we give a lower bound of $\Omega(m+n)$ on the time required to obtain a constant approximation, assuming an adjacency list representation of the network as well as the the ability to uniformly sample nodes. 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 [23, 24]. Our algorithm is randomized, and it succeeds with probability 3/5; moreover, failure is detectable, so this success probability can be amplified through repetition.

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. Our algorithms access the network structure in a very limited way: the only queries used by our algorithms are uniform vertex sampling and traversing the edges incident to a previously-accessed vertex.

To describe our approach, let us first consider the problem of finding the single node with highest influence. One strategy would be to estimate the influence of every node directly, e.g., via repeated simulation, but this is computationally expensive. Alternatively, consider the following "polling" process: select a node v uniformly at random, and determine the set of nodes that *would have influenced* v. Intuitively, if we repeat this process multiple times, and a certain node u appears often as an "influencer," then u is likely a good candidate for the most influential node. Indeed, we show that the probability a node u appears in a set of influencers is proportional to $\mathbb{E}[I(u)]$, and standard concentration bounds show that this probability can be estimated accurately with relatively few repetitions of the polling process. Moreover, it is possible to efficiently find the set of nodes that would have influenced a node v: this can be done by simulating the influence process, starting from v, in the *transpose graph* (i.e., the original network with edge directions reversed).

This motivates our algorithm, which proceeds in two steps. First, we repeatedly apply the random sampling technique described above to generate a sparse hypergraph representation of the network. 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. In the second step, we 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, note that our sampling method allows us to estimate the influence of a node, but not the marginal benefit of adding a node to a partially constructed seed set. Thus, unlike prior algorithms, we do not repeat our estimation procedure to incrementally construct a solution. Instead, we perform all of our sampling up front and then select the entire seed set using the resulting hypergraph.

Second, our algorithm has a stringent runtime constraint — we must construct our hypergraph in time $O((m + n)\epsilon^{-3} \log(n))$. To meet this bound, we must be flexible in the number of hyperedges we construct. Instead of building a certain fixed number of hyperedges, we repeatedly build edges until the *total sum of*

---

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

[2]The best implementations appear to have running time $O(mnk \log(n) \cdot \text{POLY}(\epsilon^{-1}))$ [13], though to the best of our knowledge a formal analysis of this runtime has not appeared in the literature.

all edge sizes exceeds $O((m+n)\epsilon^{-3}\log(n))$. Intuitively speaking, this works because the number of hyperedges needed to accurately estimate influence values, times the expected size of each hyperedge, is roughly constant. Indeed, we should expect to see large hyperedges only if the network contains many influential nodes, but high influence values require fewer samples to estimate accurately.

Finally, in order to prevent errors from accumulating when we apply the greedy algorithm to the hypergraph, it is important that our estimator for the influence function (i.e. total hypergraph degree) is itself a monotone submodular function.

Early Termination and Sublinear Time We next show how to modify our approximation algorithm to allow early termination, providing a tradeoff between runtime and approximation quality. Specifically, if the algorithm is allowed to run for $O(\beta(n+m)\log(n))$ steps, and is then terminated without warning, it can immediately return a solution with approximation factor $O(\beta)$. We also provide a lower bound of $\Omega(\beta(m+n)/\min\{1/\beta, k\})$ on the runtime needed to obtain an $O(\beta)$-approximation. Our algorithm is therefore nearly runtime-optimal (up to logarithmic factors) for any fixed seed size k. Our method is randomized, and it succeeds with probability $3/5$. As before, these results assume that the input network is provided in adjacency list format and an algorithm is allowed to perform uniform sampling of the nodes. Notably, our algorithm accesses the network structure in a very limited way and is directly implementable (with no additional cost) in a wide array of graph access models, including the ones used for sublinear-time and property testing algorithms on sparse graphs [36, 18] and the jump and crawl paradigm of [7].

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 the network contains many nodes with high influence, it may be that a reduction in runtime prevents us from achieving enough concentration to estimate the influence of any node. 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, is likely to have high influence.

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 by randomly selecting a singleton according to the hypergraph degree distribution. If $k > 1$ we will return a union of these two solutions. When $k = 1$ we cannot use both solutions, so we must choose; in this case, it turns out that we can determine which of the two solutions will achieve the desired approximation by examining the maximum degree in the hypergraph.

Finally, to allow early termination without warning, the algorithm can pause its hypergraph construction and compute a tentative solution at predetermined intervals (e.g., repeatedly doubling the number of steps between computations). Then, upon a request to terminate, the algorithm returns the most recent solution.

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 [21, 35, 17]. The problem of finding the most influential set of nodes to target for a diffusive process was first posed by Domingos and Richardson [16, 34]. A formal development of the IC model, along with a greedy algorithm based upon submodular maximization, was given by Kempe et al. [23]. 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 [29, 19, 28].

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 [11]. The problem of determining influence spread given a seed set in the IC model is #P-hard [12].

There has been a line of work aimed at improving the runtime of the algorithm by Kempe et al. [23]. These have focused largely on heuristics, such as assuming that all nodes have relatively low influence or that the input graph is clustered [12, 14, 25, 38], as well as empirically-motivated implementation improvements [27, 13]. One particular approach of note involves first attempting to sparsify the input graph, then estimating influence on the reduced network [14, 30]. 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 [32, 24], including those that include interations between multiple diffusive processes [20, 5]. 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, 6, 37, 2]. These problems admit local algorithms based on sampling short random walks. To the best of our understanding, these methods do not seem directly applicable to influence maximization.

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 [23], 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) directed 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 unweighted and weighted 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 $\beta \leq 1$, we say that a particular set of nodes S with $|S| \leq k$ is a β-approximation to the influence maximization problem if $\mathbb{E}_{\mathcal{G}}[I(S)] \geq \beta \cdot \max_{T:|T|=k} \mathbb{E}_{\mathcal{G}}[I(S)]$. 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.

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 discuss how to amplify this success probability in Section 3.1.

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, stochastically generates a sparse, undirected 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 hyper-

Algorithm 1 Maximize Influence

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

1: $R \leftarrow 144(m + n)\epsilon^{-3} \log(n)$
2: $\mathcal{H} \leftarrow \text{BuildHypergraph}(R)$
3: **return** $\text{BuildSeedSet}(\mathcal{H}, k)$

BuildHypergraph(R):

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

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

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

edges. 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.

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.

THEOREM 3.1. *Given any $\epsilon > 0$, 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 1. *For each subset of nodes $S \subseteq \mathcal{G}$, $\mathbb{E}_{g \sim \mathcal{G}}[I_g(S)] = n\Pr_{u,g \sim \mathcal{G}}[S \cap C_{g^T}(u) \neq \emptyset]$.*

Proof.

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

Observation 1 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 $\mathbb{E}_{\mathcal{G}}[I(\cdot)]$, 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 $R = 144(m + n)\epsilon^{-3} \log(n)$ is enough to build a sufficiently rich hypergraph structure, with high probability over the random outcomes of the influence cascade model. The idea behind the proof is to establish that $\text{OPT} \cdot \frac{m}{n}$ is an upper bound on the expected number of steps needed to build an edge, so it is unlikely that significantly fewer than $R/(\text{OPT} \cdot \frac{m}{n})$ hyperedges have been completed after R steps.

LEMMA 3.1. *Hypergraph $\mathcal{H}$ will contain at least $\frac{48n \log(n)}{\text{OPT}\epsilon^3}$ 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 $v \in Z$. Write $m_{g^T}(u)$ for the random variable indicating the number of edges that are checked as part of building the influence set Z starting at node u in g^T.

Let $X = \frac{48n \log(n)}{OPT\epsilon^3}$ 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

$$X \cdot \mathbb{E}_{u,g\sim\mathcal{G}}[1 + m_{g^T}(u)] = X + \frac{X}{n}\mathbb{E}_{g\sim\mathcal{G}}\left[\sum_u m_{g^T}(u)\right]$$

$$= X + \frac{48\log(n)}{OPT\epsilon^3}\mathbb{E}_{g\sim\mathcal{G}}\left[\sum_u m_{g^T}(u)\right]$$

$$= X + \frac{48\log(n)}{OPT\epsilon^3}\sum_{e=(v,w)\in\mathcal{G}^T}\mathbb{E}_{g\sim\mathcal{G}}\left[|\{u : v \in C_{g^T}(u)\}|\right]$$

$$= X + \frac{48\log(n)}{OPT\epsilon^3}\sum_{e=(v,w)\in\mathcal{G}^T}\mathbb{E}_{g\sim\mathcal{G}}\left[|\{u : u \in C_g(v)\}|\right]$$

$$\leq \frac{48n\log(n)}{\epsilon^3} + \frac{48\log(n)}{OPT\epsilon^3}\sum_{e=(v,w)\in\mathcal{G}^T}OPT$$

$$= \frac{48(m+n)\log(n)}{\epsilon^3}.$$

Here , the second equality (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 $R = 144(m+n)\epsilon^{-3}\log(n)$ 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 high probability. 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. Our approach is to apply concentration bounds to the random variable $deg_{\mathcal{H}}(S)$, which can be viewed as the sum of $m(\mathcal{H})$ Bernoulli random variables. However, an important subtlety is that the $m(\mathcal{H})$ is itself a random variable, determined by the stopping condition of BuildHypergraph. We must bound the correlation between $m(\mathcal{H})$ and our influence estimation. We show that the value of $m(\mathcal{H})$ is sufficiently concentrated that the resulting bias is insignificant.

LEMMA 3.2. *Suppose that $m(\mathcal{H}) \geq \frac{48n\log(n)}{OPT\epsilon^3}$. Then, for any set of nodes $S \subseteq V$, $\Pr[|\mathbb{E}_{\mathcal{G}}[I(S)] - \frac{n}{m(\mathcal{H})}deg_{\mathcal{H}}(S)| > \epsilon OPT] < \frac{1}{n^3}$, with probability taken over randomness in $\mathcal{H}$.*

Proof. We will think of $\mathcal{H}$ as being built incrementally, one edge at a time, as in the execution of BuildHypergraph. Since we will discuss the state of $\mathcal{H}$ at various points of execution, we will write M for the number of edges in $\mathcal{H}$ when BuildHypergraph terminates; we think of M as a random variable, corresponding to $m(\mathcal{H})$ in

the statement of the lemma. For a given $J \geq \frac{48n\log(n)}{OPT\epsilon^3}$, Let D_S^J denote the degree of S in $\mathcal{H}$ after J edges have been added to $\mathcal{H}$. We allow $J > M$, by considering the path of execution of BuildHypergraph had there not been a runtime bound. Thinking of D_S^J as a random variable, we have that D_S^J is the sum of J identically distributed Bernoulli random variables each with probability $\mathbb{E}_{\mathcal{G}}[I(S)]/n \geq \epsilon OPT/n$, by Observation 1. Our goal is to show that

$$(3.1)\qquad \Pr\left[|D_S^M - \mathbb{E}[D_S^M]| > \frac{\epsilon \cdot M \cdot OPT}{n}\right] < \frac{1}{n^3}.$$

We will first establish that $\Pr\left[|D_S^J - \mathbb{E}[D_S^J]| > \frac{\epsilon \cdot J \cdot OPT}{n}\right] < \frac{1}{n^{12}}$ for a fixed J. We consider two cases. First, suppose $\mathbb{E}_{\mathcal{G}}[I(S)] \geq \epsilon OPT$. In this case,

$$\mathbb{E}[D_S^J] = \mathbb{E}_{\mathcal{G}}[I(S)] \cdot \frac{J}{n} \geq \frac{\epsilon \cdot J \cdot OPT}{n} \geq 48\epsilon^{-2}\log n.$$

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

$$\Pr\left[|D_S^J - \mathbb{E}[D_S^J]| > \frac{\epsilon \cdot J \cdot OPT}{n}\right]$$
$$\leq \Pr\left[|D_S^J - \mathbb{E}[D_S^J]| < \epsilon\mathbb{E}[D_S^J]\right]$$
$$< e^{-\mathbb{E}[D_S^J]\epsilon^2/4} < e^{-12\log(n)} = \frac{1}{n^{12}}.$$

Next suppose $\mathbb{E}_{\mathcal{G}}[I(S)] < \epsilon OPT$, so $\mathbb{E}[D_S^J] < \epsilon \cdot OPT \cdot J/n$. In this case, we have $D_S^J \geq \mathbb{E}[D_S^J] - \frac{\epsilon \cdot J \cdot OPT}{n}$ surely, and the Multiplicative Chernoff bound (A.1) implies that

$$\Pr\left[D_S^J > \mathbb{E}[D_S^J] + \frac{\epsilon \cdot J \cdot OPT}{n}\right]$$
$$= \Pr\left[D_S^J > \mathbb{E}[D_S^J]\left(1 + \frac{\epsilon \cdot J \cdot OPT}{n\mathbb{E}[D_S^J]}\right)\right]$$
$$< e^{-\frac{\epsilon \cdot J \cdot OPT}{2n}} < e^{-12\log(n)} \leq \frac{1}{n^{12}}.$$

Thus, in all cases, the probability that the event of interest occurs is at most $\frac{1}{n^{12}}$.

To complete the proof, we must show that the result holds for $J = M$, noting that M is a random variable that is correlated with the event in (3.1). Suppose first that $\epsilon > n^{-4/3}$. Then since we know $M \leq R = 144(m+n)\log(n)/\epsilon^3 = o(n^7)$, we can take the union bound over all $o(n^7)$ values of J lying between $\frac{48n\log(n)}{OPT\epsilon^3}$ and R to obtain (3.1), as required.

Next suppose $\epsilon < n^{-4/3}$. We claim that M is concentrated about its expectation.

CLAIM 1. *If $\epsilon < n^{-4/3}$, then there exist $J_1, J_2 \geq \frac{48n\log(n)}{OPT\epsilon^3}$, with $J_1 \leq J_2 \leq (1 + \epsilon/2)J_1$, such that $\Pr[M \in [J_1, J_2]] > 1 - \frac{1}{n^4}$.*

Proof. Let μ be the expected size of an edge generated by BuildHypergraph, and let $\tilde{M} = R/\mu$. We claim that, with probability at least $1 - \frac{1}{n^4}$, $M \in (1 \pm \frac{\epsilon}{4})\tilde{M}$; bounding this range from below by $\frac{48n\log(n)}{\text{OPT}\epsilon^3}$ then completes the claim.

Let X denote the sum of the sizes of the first $(1 - \frac{\epsilon}{4})\tilde{M}$ edges of $\mathcal{H}$. Since each edge has size at most n, and $\mathbb{E}[X] = (1 - \frac{\epsilon}{4})\tilde{M} \cdot \mu = (1 - \frac{\epsilon}{4})R$, the Hoeffding bound implies

$$\Pr[X > R] \le \Pr\left[|X - \mathbb{E}[X]| > \frac{\epsilon R}{4}\right]$$
$$< 2e^{-\frac{R^2\epsilon^2}{16n^2(1-\epsilon)R/\mu}}$$
$$< 2e^{-\frac{R\epsilon^2}{16n^2}} < 2e^{-\log n/16\epsilon n} < \frac{2}{n^6}.$$

We therefore have $X < R$, which implies $M > (1 - \frac{\epsilon}{4})\tilde{M}$, with probability at least $1 - \frac{2}{n^6}$. A similar argument yields $M < (1 + \frac{\epsilon}{4})\tilde{M}$ with probability at least $1 - \frac{2}{n^6}$. We conclude (via union bound) that $M \in (1 \pm \frac{\epsilon}{4})\tilde{M}$ with probability at least $1 - \frac{4}{n^6}$, which proves the claim.

Condition on the event that $M \in [J_1, J_2]$, which has probability at least $1 - \frac{1}{n^4}$. We know that with probability at least $1 - \frac{1}{n^4}$, $|D_S^{J_2} - \mathbb{E}[D_S^{J_2}]| < \frac{\epsilon \cdot J_2 \cdot \text{OPT}}{2n}$; we will condition on this event as well. Since D_S^M is dominated by $D_S^{J_2}$, and since $J_2 < (1 + \epsilon/2)J_1 \le (1 + \epsilon/2)M$, we conclude that

$$D_S^M \le D_S^{J_2} < \mathbb{E}[D_S^{J_2}] + \frac{\epsilon\text{OPT}}{2n}J_2$$
$$= \mathbb{E}[D_S] \cdot \frac{J_2}{M} + \frac{\epsilon\text{OPT}}{2n}J_2$$
$$< \mathbb{E}[D_S] + \frac{\epsilon}{2}\mathbb{E}[D_S] + \frac{\epsilon\text{OPT}}{2n}M$$
$$< \mathbb{E}[D_S] + \frac{\epsilon\text{OPT}}{n}M$$

as required. Similarly, the fact that D_S dominates $D_S^{J_1}$ implies $D_S > \mathbb{E}[D_S] - \frac{\epsilon\text{OPT}}{n}M$, conditioning on an event with probability at least $1 - \frac{1}{n^4}$, as required. Taking the union bound over the complement of the conditioned events, we conclude that the unconditional probability of (3.1) is at most $\frac{3}{n^4} < \frac{1}{n^3}$ 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 3.3. *Choose $\delta > 0$ and suppose that $f: 2^V \to \mathbb{R}_{\ge 0}$ 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 $|S| \le k$, $\Pr_{\hat{f} \sim D}[|f(S) - \hat{f}(S)| > \delta] < 1/n^3$. If we write $S_{\hat{f}}$ for the set returned by the greedy algorithm on input $\hat{f}$, then*

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

Proof. Lemma 3.3 Choose $S^* \in \text{argmax}_{|S|=k}\{f(S)\}$. With probability at least $1 - 1/n^3$, $\hat{f}(S^*) \ge f(S^*) - \delta$. So, in particular, $\max_{|S|=k} \hat{f}(S) \ge f(S^*) - \delta$.

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

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

conditioning on an event of probability $1 - 1/n$. $\quad\square$

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

Proof. [Proof of Theorem 3.1] Lemma 3.1 and 3.2 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 3.3 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 $\delta = \epsilon\text{OPT}$. Lemma 3.3 implies that, with probability at least $1 - \frac{1}{n}$, the greedy algorithm applied to $\mathcal{H}$ returns a set S with $\mathbb{E}_{\mathcal{G}}[I(S)] \ge (1 - 1/e)\text{OPT} - 2\epsilon\text{OPT} = (1 - 1/e - 2\epsilon)\text{OPT}$. 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 Build-Hypergraph 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 3.1: 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 3.2 becomes dominant).

We next note that, for any $\ell > 1$, the error bound in Lemma 3.2 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

We now describe a modified algorithm that provides a tradeoff between runtime and approximation quality. For an an arbitrary $\beta < 1$, our algorithm will obtain an $O(\beta)$-approximation to the influence maximization problem, in time $O(\beta(n+m) \log(n))$, with probability at least $3/5$. In Section 4.1 we describe an implementation of this algorithm that supports termination after an arbitrary number of steps, rather than being given the value of β in advance.

Our algorithm is listed as Algorithm 2. 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 to construct hypergraph $\mathcal{H}$. 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 fewer samples: if β is sufficiently small, we may not have enough data to guess at 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 sampling approach would fail.

If $k > 1$, we can combine these two approaches by returning a union of vertices selected according to each procedure. If $k = 1$, we must choose which approach to apply. However, in this case, there is a simple way to determine whether we have obtained enough samples that BuildSeedSet returns an acceptable solution: check whether the maximum degree in the hypergraph is sufficiently high.

THEOREM 4.1. *For any $\beta < 1$, Algorithm 2 returns, with probability of at least $3/5$, a node with expected influence at least $min\{\frac{1}{4}, \beta\} \cdot OPT$. Its runtime is $O(\beta(n + m) \log(n))$.*

Our proof of Theorem 4.1 proceeds via two cases, depending on whether $\mathcal{H}$ has many or few edges as a function of OPT. The precise number of edges we require involves the constant $C = 48 \cdot 6^3$ (from the definition of Algorithm 2), which we have not tried to

Algorithm 2 Runtime-Flexible Influence Maximization

Define: $C = 48 \cdot 6^3$
Require: Approximation parameter $\beta < 1$, directed weighted graph $\mathcal{G}$.

1: $R \leftarrow \beta \cdot 144C \cdot (n + m) \log(n)$
2: $\mathcal{H} \leftarrow \text{BuildHypergraph}(R)$
3: Choose $v \in V$ with probability proportional to degree in $\mathcal{H}$
4: **if** $k > 1$ **then**
5:     $S \leftarrow \text{BuildSeedSet}(\mathcal{H}, k-1)$
6:     **return** $S \cup \{v\}$
7: **else**
8:     $S \leftarrow \text{BuildSeedSet}(\mathcal{H}, 1)$
9:     **if** $\max_u\{deg_{\mathcal{H}}(u)\} > 2C \log n$ **then return** S
10:     **else return** $\{v\}$

---

optimize. We first show that, subject to $\mathcal{H}$ having many edges, set S from line 5 or 8 (corresponding to $k > 1$ and $k = 1$, respectively) is likely to have high influence. This follows the analysis from Theorem 3.1.

LEMMA 4.1. *Suppose that $m(\mathcal{H}) \geq \frac{Cn\log(n)}{\text{OPT}}$. Then, with probability at least $1 - \frac{1}{n}$, set S satisfies $\mathbb{E}_{\mathcal{G}}[I(S)] \geq \frac{1}{4}\text{OPT}$, with probability taken over randomness in $\mathcal{H}$.*

Proof. Recall $C = 48 \cdot 6^3$. First suppose $k = 1$, so S is as defined on line 8. If we apply Lemma 3.2 with $\epsilon = \frac{1}{6}$, followed by the analysis of BuildSeedSet$(\mathcal{H}, k)$ from the proof of Theorem 3.1, we get that $\mathbb{E}_{\mathcal{G}}[I(S)] \geq \frac{1}{2}\text{OPT}$, as required. For $k > 1$, applying Lemma 3.2 with $\epsilon = \frac{1}{6}$ yields instead that $\mathbb{E}_{\mathcal{G}}[I(S)] \geq \frac{1}{2}\text{OPT}_{k-1}$, where OPT_{k-1} is the maximum influence over sets of size at most $k - 1$. But now, by submodularity, $\frac{1}{2}\text{OPT}_{k-1} \geq \frac{1}{2}(\frac{k-1}{k})\text{OPT} \geq \frac{1}{4}\text{OPT}$, as required. $\square$

Note that β does not appear explicitly in the statement of Lemma 4.1. The (implicit) role of β in Lemma 4.1 is that as β becomes small, 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 3 is likely to have high influence. This follows because, in a small number of edges, we do not expect to see many nodes with low influence. Since we see a large number of nodes in total, we conclude that most of them must have high influence.

LEMMA 4.2. *Suppose that $m(\mathcal{H}) < \frac{4Cn\log(n)}{\text{OPT}}$. Then, with probability at least $2/3$, node v (from line 3 of Algorithm 2) satisfies $\mathbb{E}_{\mathcal{G}}[I(v)] \geq \beta \cdot \text{OPT}$, with probability taken over randomness in $\mathcal{H}$.*

Proof. Recall $C = (48 \cdot 6^3)$. Let random variable X denote the number of times that a node with influence at most $\beta \cdot \text{OPT}$ was added to a hyperedge of $\mathcal{H}$. Since $\mathcal{H}$ has fewer than $\frac{4Cn\log(n)}{\text{OPT}}$ edges, the expected value of X is at most

$$E[X] \leq \frac{4Cn\log(n)}{\text{OPT}} \sum_{u \in V} \frac{1}{n} \min\{\mathbb{E}_{\mathcal{G}}[I(u)], \beta \cdot \text{OPT}\}$$
$$\leq 4C\beta n \log(n).$$

Markov inequality then gives that $\Pr[X > 24C\beta n \log(n)] < 1/6$. Conditioning on this event, we have that at most $24C\beta n \log(n)$ of the nodes touched by BuildHypergraph have influence less than $\beta \cdot \text{OPT}$. Since at least $(144C)\beta n \log(n)$ nodes were touched in total, the probability that node v from line 4 has influence less than $\beta \cdot \text{OPT}$ is at most $1/6$. The union bound then allows us to conclude that v has $\mathbb{E}[I(v)] \geq \beta \cdot \text{OPT}$ with probability at least $1 - (1/6 + 1/6) \geq 2/3$. $\square$

For the case $k = 1$, the algorithm chooses between returning S and returning $\{v\}$, based on the maximum degree in $\mathcal{H}$. The following lemma motivates this choice. The proof follows from an application of concentration bounds: if a node is present in $O(\log n)$ hyperedges, then with high probability we have obtained enough samples to accurately estimate its influence.

LEMMA 4.3. *If $k = 1$ then the following is true with probability at least $1 - \frac{2}{n}$. If $\max_u\{deg_{\mathcal{H}}(u)\} > 2C \log n$ then $m(\mathcal{H}) > \frac{Cn\log(n)}{\text{OPT}}$. Otherwise, $m(\mathcal{H}) < \frac{4Cn\log(n)}{\text{OPT}}$.*

Proof. As in the proof of Lemma 3.2, we will think of $\mathcal{H}$ as being built incrementally edge by edge, and we will let D_w^J denote the degree of w in $\mathcal{H}$ after J edges have been added. Then, for any fixed J, D_w^J is precisely the sum of J Bernoulli random variables, each with expectation $\frac{1}{n}\mathbb{E}_{\mathcal{G}}[I(w)]$, and hence $\mathbb{E}[D_w^J] = \frac{J}{n}\mathbb{E}_{\mathcal{G}}[I(w)]$.

Let J_1 be the minimal J such that $\mathbb{E}[D_w^{J_1}] > 4C \log n$. Chernoff bounds (Lemma A.1) imply

$$\Pr[D_w^{J_1} \leq 2C \log n] < e^{-2\log n} = 1/n^2.$$

Suppose that this event does not occur. Then for any $J \geq J_1$, we have $D_w^J > D_w^{J_1} > 2C \log n$. In particular, if $\mathbb{E}[D_w^{m(\mathcal{H})}] > 4C \log n$, we must have $m(\mathcal{H}) \geq J_1$, and hence $D_w^{m(\mathcal{H})} > 2C \log n$.

Let J_2 be the maximal J such that $\mathbb{E}[D_w^{J_2}] < C \log n$. Chernoff bounds (Lemma A.1) again imply that

$$\Pr[D_w^{J_2} \geq 2C \log n] < e^{-2\log n} = 1/n^2.$$

Suppose that this event does not occur. Then for any $J \leq J_2$, we have $D_w^J < D_w^{J_2} < 2C \log n$. In particular,

for any w such that $\mathbb{E}[D_w^{m(\mathcal{H})}] < C \log n$, we must have $m(\mathcal{H}) \leq J_2$, and hence $D_w^{m(\mathcal{H})} < 2C \log n$.

Taking the union bound over all w, we conclude that with probability at least $1 - 2/n$, only w for which $\mathbb{E}[D_w^{m(\mathcal{H})}] > C \log n$, and every w with $\mathbb{E}[D_w^{m(\mathcal{H})}] > 4C \log n$, will have $D_w^{m(\mathcal{H})} \geq 2C \log n$. We will condition on this event for the remainder of the proof.

Suppose that $\max_w D_w^{m(\mathcal{H})} < 2C \log n$. Then we have that $\max_w \mathbb{E}[D_w^{m(\mathcal{H})}] < 4C \log n$. Since $\max_w \mathbb{E}[D_w^{m(\mathcal{H})}] = \max_w \frac{1}{n} m(\mathcal{H}) \mathbb{E}_{\mathcal{G}}[I(w)] = \frac{1}{n} m(\mathcal{H}) \cdot OPT$, we conclude $m(\mathcal{H}) < \frac{4Cn \log(n)}{OPT}$ as required. Next suppose that $\max_w D_w^{m(\mathcal{H})} > 2C \log n$. We then have that $\max_w \mathbb{E}[D_w^{m(\mathcal{H})}] > C \log n$. Since, again, $\max_w \mathbb{E}[D_w^{m(\mathcal{H})}] = \frac{1}{n} m(\mathcal{H}) \cdot OPT$, we conclude $m(\mathcal{H}) > \frac{Cn \log(n)}{OPT}$ as required.

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

Proof. [Proof of Theorem 4.1] Lemma 4.1 and Lemma 4.2 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 $\min\{\frac{1}{4}, \beta\} \cdot OPT$, and therefore $S \cup \{v\}$ does as well. If $k > 1$ then we return $S \cup \{v\}$ and we are done. Otherwise, Lemma 4.3 implies that if we return set S then the influence of S is at least $OPT/4$ (by Lemma 4.1), and if we return set v then the expected influence of v is at least $\beta \cdot OPT$ (by Lemma 4.2). Thus, in all cases, we return a set of influence at least $\min\{\frac{1}{4}, \beta\} \cdot OPT$. The required bound on the runtime of Algorithm 2 follows directly from the value of R on line 1, as in the proof of Theorem 3.1.

4.1 Dynamic Runtime Algorithm 2 assumes that the desired approximation factor, β, is provided as a parameter to the problem. We note that a slight modification to the algorithm removes the requirement that β be specified in advance. That is, we obtain an algorithm that can be terminated without warning, say after $O(\gamma \cdot (n + m) \log(n))$ steps for some $\gamma \leq 1$, at which point it immediately returns a solution that is an $O(\gamma)$ approximation with probability at least $\frac{3}{5}$. To achieve this, we execute Algorithm 2 as though $\beta = 1$, but then modify BuildHypergraph so that, for each $i \geq 1$, we pause the creation of hypergraph $\mathcal{H}$ after 2^i steps and complete the algorithm using the current hypergraph, which takes time at most $O(2^i)$. Once this is done, we save the resulting solution and resume the creation of the hypergraph until the next power of 2. When the algorithm is terminated, we return the most recently-computed solution; this corresponds to a solution for a hypergraph built using at least half

of the total steps taken by the algorithm at the time of termination. Theorem 4.1 then implies that this solution has approximation $O(\gamma)$ if termination occurs after $O(\gamma \cdot (n + m) \log(n))$ steps.

4.2 A Lower Bound We provide a lower bound on the time it takes for any algorithm, equipped with uniform node sampling, to compute a β-approximation for the maximum expected influence problem under the adjacency list network representation. In particular, for any given budget k, at least $\Omega(\beta n)$ queries are required to obtain approximation factor β with fixed probability.

THEOREM 4.2. *Let $0 < \epsilon < \frac{1}{10e}$, $\beta \leq 1$ be given. Any randomized algorithm for the maximum influence problem that has runtime of $\frac{\beta(m+n)}{24 \min\{k, 1/\beta\}}$ cannot return, with probability at least $1 - \frac{1}{e} - \epsilon$, a set of nodes with approximation ratio better than β.*

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

The proof will use Yao's Minimax Principle for the performance of Las Vegas (LV) randomized algorithms on a family of inputs [39]. The lemma states that the least expected cost of deterministic LV algorithms on a distribution over a family inputs is a lower bound on the expected cost of the optimal randomized LV algorithm over that family of inputs. Define the cost of the algorithm as 0 if it returns a set nodes with approximation ratio better than β and 1 otherwise. As the cost of an algorithm equals its probability of failure, we can think of it as a LV algorithm.

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

We now consider the behavior of a deterministic algorithm A with respect to the uniform distribution on the constructed family of inputs. For a given value T the graph would be made from k components of size $2T$ and $n - 2kT$ singleton components (recall that $2kT = 2k/\beta \leq n$). If algorithm A returns nodes from ℓ of the k components of size $2T$, it achieves a total influence of $2\ell T + (k - \ell)$. Thus, to attain approximation factor better than $\beta = \frac{1}{T}$, we must have $2\ell T + (k - \ell) \geq \frac{1}{T} 2kT$, which implies $\ell \geq \frac{k}{2T - 1}$ for any $T > 1$.

Suppose $k > 12T$. The condition $\ell \geq \frac{k}{2T-1}$ implies that at least $\frac{k}{2T-1}$ of the large components must be queried by the algorithm, where each random query

has probability $\frac{2kT}{n}$ of hitting a large component. If the algorithm makes fewer than $\frac{n}{12T^2}$ queries, then the expected number of components hit is $\frac{n}{12T^2} \cdot \frac{2kT}{n} = \frac{k}{6T}$. The Multiplicative Chernoff bound (Lemma A.1, part 3) then imply that the probability hitting more than $\frac{k}{2T}$ components is no more than $e^{-\frac{k}{6T} \cdot 2/3} \leq \frac{1}{e^{4/3}} < 1 - \frac{1}{e} - \epsilon$, a contradiction.

If $k \leq 12T$ then we need that $\ell \geq 1$, which occurs only if the algorithm queries at least one of the kT vertices in the large components. With $\frac{n}{2kT}$ 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 $\frac{n}{12T \min\{k,T\}}$ queries are necessary to obtain approximation factor better than $\beta = \frac{1}{T}$ with probability at least $1 - \frac{1}{e} - \epsilon$, as required.

By Yao's Minimax Principle this gives a lower bound of $\Omega(\frac{nd}{24T \min\{k,T\}})$ on the expected performance of any randomized algorithm, on at least one of the inputs.

Finally, the construction can be modified to apply to non-sparse networks. 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, but increases the time to decide if 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 $\frac{nd}{24T \min\{k,T\}}$ on the expected performance of A on a distribution of networks with $m = nd$ edges.

Discussion: The lower bound construction of Theorem 4.2 is tailored to the query model considered in this paper. In particular, we do not 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 regular. $\square$

A Multiplicative Chernoff Bound

For reference, we now provide the statement of the Chernoff bounds used throughout this paper.

LEMMA A.1. *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 $\lambda \geq 1$: $\Pr[X > (1 + \lambda)\mu n] < \exp(-\mu n \lambda/3)$.*

References

[1] ANDERSEN, R., BORGS, C., CHAYES, J., HOPCRAFT, J., MIRROKNI, V. S., AND TENG, S.-H. 2007. Local computation of pagerank contributions. In *WAW*. 150–165.

[2] ANDERSEN, R., CHUNG, F., AND LANG, K. 2006. Local graph partitioning using pagerank vectors. In *FOCS*. 475–486.

[3] BAKSHY, E., KARRER, B., AND ADAMIC, L. A. 2009. Social influence and the diffusion of user-created content. In *ACM EC*. 325–334.

[4] BEN-ZWI, O., HERMELIN, D., LOKSHTANOV, D., AND NEWMAN, I. 2011. Treewidth governs the complexity of target set selection. *Discrete Opt. 8*, 1, 87–96.

[5] BHARATHI, S., KEMPE, D., AND SALEK, M. 2007. Competitive influence maximization in social networks. In *WINE*. 306–311.

[6] BORGS, C., BRAUTBAR, M., CHAYES, J. T., AND TENG, S.-H. 2012. A sublinear time algorithm for pagerank computations. In *WAW*. 41–53.

[7] BRAUTBAR, M. AND KEARNS, M. 2010. Local algorithms for finding interesting individuals in large networks. In *ICS*. 188–199.

[8] BROWN, J. J. AND REINGEN, P. H. 1987. Social ties and word of mouth referral behavior. *Journal of Consumer Research 14,* 3, 350–362.

[9] CENTOLA, D. AND MACY, M. 2007. Complex contagions and the weakness of long ties. *American Journal of Sociology 113,* 3, 702–734.

[10] CHA, M., MISLOVE, A., AND GUMMADI, P. K. 2009. A measurement-driven analysis of information propagation in the flickr social network. In *WWW*. 721–730.

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

[12] CHEN, W., WANG, C., AND WANG, Y. 2010a. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *KDD*. 1029–1038.

[13] CHEN, W., WANG, Y., AND YANG, S. 2009. Efficient influence maximization in social networks. In *KDD*. 199–208.

[14] CHEN, W., YUAN, Y., AND ZHANG, L. 2010b. Scalable influence maximization in social networks under the linear threshold model. In *ICDM*. 88–97.

[15] DODDS, P. AND WATTS, D. 2007. Universal behavior in a generalized model of contagion. *Phys Rev Lett 92,* 21, 218701.

[16] DOMINGOS, P. AND RICHARDSON, M. 2001. Mining the network value of customers. In *KDD*. 57–66.

[17] GOLDENBERG, J., LIBAI, B., AND MULLE, E. 2001. Talk of the network: A complex systems look at the underlying process of word-of-mouth. *Mark. Let.*, 221–223.

[18] GOLDREICH, O. 2010. Introduction to testing graph properties. In *Property Testing*. 105–141.

[19] GOMEZ-RODRIGUEZ, M., LESKOVEC, J., AND KRAUSE, A. 2012. Inferring networks of diffusion and influence. *TKDD 5,* 4, 21.

[20] GOYAL, S. AND KEARNS, M. 2012. Competitive contagion in networks. In *STOC*. 759–774.

[21] GRANOVETTER, M. 1978. Threshold models of collective behavior. *American Journal of Sociology 83,* 1420–1443.

[22] JIANG, Q., SONG, G., CONG, G., WANG, Y., SI, W., AND XIE, K. 2011. Simulated annealing based influence maximization in social networks. In *AAAI*.

[23] KEMPE, D., KLEINBERG, J., AND TARDOS, E. 2003. Maximizing the spread of influence through a social network. In *KDD*. 137–146.

[24] KEMPE, D., KLEINBERG, J. M., AND TARDOS, É. 2005. Influential nodes in a diffusion model for social networks. In *ICALP*. 1127–1138.

[25] KIMURA, M. AND SAITO, K. 2006. Tractable models for information diffusion in social networks. In *PKDD*. 259–271.

[26] LESKOVEC, J., ADAMIC, L. A., AND HUBERMAN, B. A. 2007a. The dynamics of viral marketing. *TWEB 1,* 1.

[27] LESKOVEC, J., KRAUSE, A., GUESTRIN, C., FALOUTSOS, C., VANBRIESEN, J. M., AND GLANCE, N. S. 2007b. Cost-effective outbreak detection in networks. In *KDD*. 420–429.

[28] LESKOVEC, J., McGLOHON, M., FALOUTSOS, C., GLANCE, N. S., AND HURST, M. 2007c. Patterns of cascading behavior in large blog graphs. In *SDM*.

[29] LIBEN-NOWELL, D. AND KLEINBERG, J. 2008. Tracing information flow on a global scale using internet chain-letter data. *PNAS 105,* 12, 4633–4638.

[30] MATHIOUDAKIS, M., BONCHI, F., CASTILLO, C., GIONIS, A., AND UKKONEN, A. 2011. Sparsification of influence networks. In *KDD*. 529–537.

[31] MORRIS, S. 2000. Contagion. *Review of Economic Studies 67,* 57–78.

[32] MOSSEL, E. AND ROCH, S. 2007. On the submodularity of influence in social networks. In *STOC*. 128–134.

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

[34] RICHARDSON, M. AND DOMINGOS, P. 2002. Mining knowledge-sharing sites for viral marketing. In *KDD*. 61–70.

[35] ROGERS, E. 2003. *Diffusion of Innovations* 5th Ed. Free Press.

[36] RUBINFELD, R. AND SHAPIRA, A. 2011. Sublinear time algorithms. *SIAM Journal on Discrete Math 25,* 1562–1588.

[37] SPIELMAN, D. A. AND TENG, S.-H. 2004. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *STOC*. 81–90.

[38] WANG, Y., CONG, G., SONG, G., AND XIE, K. 2010. Community-based greedy algorithm for mining top-k influential nodes in mobile social networks. In *KDD*. 1039–1048.

[39] YAO, A. C.-C. 1977. Probabilistic computations: Toward a unified measure of complexity (extended abstract). In *FOCS*. 222–227.