TipTop: (Almost) Exact Solutions for Influence Maximization in Billion-Scale Networks

Xiang Li, *Member, IEEE*, J. David Smith, Thang N. Dinh, *Member, IEEE*, and My T. Thai, *Senior Member, IEEE*

***Abstract*— In this paper, we study the cost-aware target viral marketing (CTVM) problem, a generalization of influence maximization. CTVM asks for the most cost-effective users to influence the most relevant users. In contrast to the vast literature, we attempt to offer exact solutions. As the problem is NP-hard, thus, exact solutions are intractable, we propose TipTop, a $(1-\epsilon)$-optimal solution for arbitrary $\epsilon > 0$ that scales to very large networks, such as Twitter. At the heart of TipTop lies an innovative technique that reduces the number of samples as much as possible. This allows us to exactly solve CTVM on a much smaller space of generated samples using integer programming. Furthermore, TipTop lends a tool for researchers to benchmark their solutions against the optimal one in large-scale networks, which is currently not available.**

***Index Terms*— Viral marketing, influence maximization, algorithms, online social networks, optimization.**

I. Introduction

With the recent development, Online Social Networks (OSNs) have become one of the most effective platforms for marketing and advertising. Through "word-of-mouth" exchanges, so-called viral marketing, the influence and product adoption can spread from few key users to billions of users in the network. To identify these key users, Influence Maximization (IM) problem, which asks for a set of k seed users that maximizes the expected number of influenced nodes, has been studied extensively [2]–[7] (and references therein). Taking into account both arbitrary cost for selecting a node and arbitrary benefit for influencing a node, a generalized problem, Cost-aware Targeted Viral Marketing (CTVM), has been introduced recently [8]. Given a budget κ, CTVM asks to find a seed set S with the total cost at most κ such as to maximize the expected total benefit over the influenced nodes.

Manuscript received August 25, 2017; revised August 25, 2018; accepted January 9, 2019; approved by IEEE/ACM Transactions on Networking Editor A. Wierman. Date of publication March 6, 2019; date of current version April 16, 2019. This work was supported in part by the NSF under Grant CNS-1443905 and Grant EFRI 1441231 and in part by the Defense Threat Reduction Agency under Grant HDTRA1-14-1-0055. The preliminary version of this paper has appeared in [1]. *(Corresponding author: My T. Thai.)*

X. Li is with the Department of Computer Engineering, Santa Clara University, Santa Clara, CA 95053 USA (e-mail: xli8@scu.edu).

J. D. Smith is with the Computer and Information Science and Engineering Department, University of Florida, Gainesville, FL 32601 USA (e-mail: jdsmith@cise.ufl.edu).

T. N. Dinh is with the Computer Science Department, Virginia Commonwealth University, Richmond, VA 23284 USA (e-mail: tndinh@vcu.edu).

M. T. Thai is with the Division of Algorithms and Technologies for Networks Analysis & Faculty of Information Technology, Ton Duc Thang University, Ho Chi Minh City, Vietnam, and also with the Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL 32601 USA (e-mail: thaitramy@tdt.edu.vn; mythai@cise.ufl.edu).

Digital Object Identifier 10.1109/TNET.2019.2898413

Despite a great amount of works [2]–[11], none of these attempts to solve IM or CTVM exactly. The lack of such a solution makes it challenging to evaluate the performance of existing solutions, such as IMM [6] and SSA [7] algorithms, in real-world datasets, against optimal solutions. Despite the fact that these algorithms have a theoretical performance guarantee of $(1-1/e-\epsilon)$ in the worst case, one can always ask: how well do these algorithms actually perform on the billion-scale OSNs? This question has remained unanswered til now.

Obtaining exact solutions to CTVM (and thus to IM) indeed is very challenging. Due to the nature of the problem, stochastic programming is a viable approach for optimization under uncertainty when the probability distribution governs the data is given [12]. However traditional stochastic programming-based solutions to various problems in NP-hard class are only for small networks with a few hundreds nodes [12]. Thus directly applying existing techniques to IM and CTVM is not suitable because OSNs consist of millions of users and billions of edges. Furthermore, the theory developed to assess the solution quality such as those in [12], [13], and the references therein, only provide approximate confidence interval. Therefore, sufficiently large samples are needed to justify the quality assessment. This requires us to develop novel stochastic programming techniques to optimally solve CTVM.

In this paper, we provide the first (almost) exact solutions for CTVM with an approximation ratio of $(1-\epsilon)$. To tackle the above challenges, we develop two innovative techniques: 1) Reduce the number of samples as much as possible so that the stochastic programming can be solved in a short time. This requires us to tightly bound the number of samples needed to generate a candidate solution. 2) Develop novel computational method to assess the solution quality with just enough samples, where the quality requirement is given a priori. These results cross the barriers in stochastic programming theory where solving stochastic programming on large-scale networks had been thought impractical. Our contributions are summarized as follows:

- Design two exact solutions, namely T-EXACT and E-EXACT, to CTVM using two-stage stochastic programming which utilizes the sample average approximation method to reduce the number of realizations. Using T-EXACT and E-EXACT, we illustrate that traditional stochastic programming techniques badly suffer the scalability issue.
- Develop an (almost) optimal algorithm to CTVM with a performance ratio of $(1-\epsilon)$: The Tiny Integer Program

with Theoretically OPtimal results (TipTop). Being able to obtain the optimal solution, TipTop is used as a benchmark to evaluate the absolute performance of existing solutions in billion-scale OSNs.

- Conduct extensive experiments confirming that the theoretical performance of TipTop is attained in practice. Our experiments show that it is feasible to compute 98% optimal solutions to CTVM on billion-scale OSNs. These experiments confirm that our sampling reductions are significant in practice, by a factor of 10^3 on average.

Organization: Section II briefly discusses the related work and the Reverse Influence Sampling (RIS). In Section III, we present the network model, propagation models, and the problem definition. Section IV presents our EXACT algorithms for CTVM. Our main contribution, TipTop is introduced in Section V. We analyze TipTop approximation factor in Section VI. Experimental results on real social networks are shown in Section VII. And finally Section VIII concludes the paper.

II. Related Work and Reverse Influence Sampling

Influence Maximization: Kempe *et al.* [9] formulated viral marketing as the IM optimization problem, focused on two fundamental cascade models, Linear Threshold (LT) and Independent Cascade (IC) models. They showed the problem to be NP-complete and devised an $(1 - 1/e - \epsilon)$ approximation algorithm. In addition, IM cannot be approximated within a factor $(1 - \frac{1}{e} + \epsilon)$ [14] under a typical complexity assumption. Computing the exact influence is shown to be #P-hard [3].

Following [9], a series of work have been proposed, focused on improving the time complexity [2]–[5], [15]–[19]. All of these work retain the ratio of $(1-1/e-\epsilon)$. Their major bottleneck is the inefficiency in estimating the influence spread, thus restraining them from being able to run on large networks.

Reverse Influence Sampling (RIS): Borgs *et al.* [10] have introduced a novel sampling approach, RIS, which is a foundation for the later works. Briefly, RIS captures the influence landscape of $G = (V, E, p)$ through generating a hypergraph $\mathcal{H} = (V, \{\mathcal{E}_1, \mathcal{E}_2, \ldots\})$. Each hyperedge $\mathcal{E}_j \in \mathcal{H}$ is a subset of nodes in V and constructed as follows: 1) selecting a random node $v \in V$ 2) generating a sample graph $g \sqsubseteq G$ and 3) returning $\mathcal{E}_j$ as the set of nodes that can reach v in g. Observe that $\mathcal{E}_j$ contains the nodes that can influence its source v. If we generate multiple random hyperedges, influential nodes will likely appear more often in the hyperedges. Thus a seed set S that *covers* most of the hyperedges will likely maximize the influence spread. Here S covers a hyperedge $\mathcal{E}_j$, if $S \cap \mathcal{E}_j \neq \emptyset$. Therefore, IM can be solved using the following framework. 1) Generate multiple random hyperedges from G. 2) Use the greedy algorithm for the Max-coverage problem [20] to find S that covers the maximum number of hyperedges and return S as the solution. The core issue in applying the above framework is that: How many hyperedges are sufficient to provide a good approximation solution?

Based on RIS, Borgs *et al.* [10] presented an $O(kl^2(m + n)\log^2 n/\epsilon^3)$ time algorithm for IM under IC model. It returns a $(1 - 1/e - \epsilon)$-approximate ratio with probability at least $1 - n^{-l}$. In practice, the proposed algorithm is, however, less than satisfactory due to the rather large hidden constants. In a sequential work, Tang et al. [6] reduced the running time to $O((k+l)(m+n)\log n/\epsilon^2)$ and showed that their algorithm is efficient in billion-scale networks. Nguyen et al. [7] proposed SSA/DSSA algorithms to further reduce the running time up to orders of magnitudes. The algorithm keeps generating samples and stops at exponential check points to verify (stare) if there is adequate statistical evidence on the solution quality for termination. Huang et al. showed gaps in SSA/D-SSA [21] and propose the fixes for SSA. Independently, the authors of SSA/D-SSA provided the fixes for both SSA and D-SSA in [22] with the summary of changes in [23]. However, SSA does not put an effort in minimizing the number of samples at the stopping point to verify the candidate solution, which is needed to solve the Integer Programming (IP). All of these works have a $(1 - 1/e - \epsilon)$-approximation ratio.

Generalization: In generalizing IM, Nguyen and Zheng [11] investigated the BIM problem in which each node can have an arbitrary selecting cost. They proposed a $(1 - 1/\sqrt{e} - \epsilon)$ approximation algorithm (called BIM) based on a greedy algorithm for Budgeted Max-Coverage in [20] and two heuristics. However, none of the proposed algorithms can handle billion-scale networks. Recently, Nguyen *et al.* introduced CTVM and presented a scalable $(1 - 1/\sqrt{e} - \epsilon)$ algorithm [8]. They also showed that straightforward adaption of the methods in [5], [6], and [10] for CTVM can incur an excessive number of samples, thus, are not efficient enough for large networks.

III. Models and Problem Definitions

Let $\mathcal{G} = (V, E, c, b, \mathcal{D})$ be a network with a node set V and a directed edge set E, with $|V| = n$ and $|E| = m$. Each node $u \in V$ has a selecting cost $c_u \geq 0$, also written $c(u)$, and a benefit $b(u)$ if u is influenced.[1] Each directed edge $(u, v) \in E$ is associated with an influence probability $p_{uv} \in [0, 1]$.

The diffusion of information in $\mathcal{G}$ is captured through a probability space $\mathcal{D}$ that associates each possible cascade graph $g = (V, E_s), E_s \subseteq E$ with a probability. Each cascade graph g is also called a *realization* or a *sample graph* of $\mathcal{G}$. An edge $(u, v) \in E_s$ in g implies that u can activate v in that graph.

Given a seed set $S \subset V$, the number of nodes get influenced by S within a sample graph g is defined as the number of nodes reachable from S in g, and denoted by $R(g, S)$. The *influence spread* of S in $\mathcal{G}$ is, similarly, defined as the expected influence of S over all possible realization graphs in $\mathcal{D}$. Mathematically, define $\mathcal{D} = (\Omega, \mathcal{F}, P)$ where $\Omega = \{g = (V, E_s) | E_s \subseteq E\}$ is the set of all possible graph samples of $\mathcal{G}$, $\mathcal{F} = 2^\Omega$, and $P : \mathcal{F} \to [0, 1]$, a probability measure.

Let G denote a *random graph* defined over $\mathcal{D}$. The influence spread of the seed set S is

$$\mathbb{I}(S) = \mathbb{E}[R(G, S)] = \sum_{g \in \Omega} P(g)|R(g, S)|, \qquad (1)$$

[1]The cost of node u, c_u, can be estimated proportionally to the centrality of u (how important the respective person is), e.g., out-degree of u [11]. Additionally, the node benefit $b(u)$ refers to the gain of influencing node u, e.g., 1 for each node in our targeted group and 0 outside [24].

where $R(g, S)$ denotes the set of nodes reachable from S within g.

For example, we consider the popular Independent Cascade (IC) model [9]. In IC, the influence propagation happens in round $t = 1, 2, 3, \ldots$. At round 1, nodes in S are *activated* and the other nodes are *inactive*. The cost of activating S is given $c(S) = \sum_{u \in S} c_u$. At round $t > 1$, each newly activated node u will independently activate its neighbor v with a probability p_{uv}. Once a node becomes activated, it remains activated in all subsequent rounds. The influence propagation stops when no more nodes are activated.

For IC, the probability mass function for each sample graph $g = (V, E_s)$ is

$$P(g) = \mathsf{Pr}[G = g] = \prod_{e \in E_s} p_e \prod_{e \in E \setminus E_s} (1 - p_e).$$

Similarly, the *benefit* of S is defined as the expected total benefit over all influenced nodes, i.e.,

$$\mathbb{B}(S) = \mathbb{E}[B(G, S)] = \sum_{g \in \Omega} P(g) B(g, S), \tag{2}$$

where $B(g, S) = \sum_{u \in R(g,S)} b(u)$ is the benefit of selecting S with respect to (w.r.t.) graph sample g.

Without loss of generality (w.l.o.g.), we assume that the benefit of the nodes are normalized so that:

$$\sigma_B = \sum_{u \in V} b(u) = n. \tag{3}$$

This is compatible with the uniform benefit case in the IM problem in which each node has a same benefit one and $B(g, S) = R(g, S)$.

We are now ready to define the CTVM problem as follows.

Definition 1 (Cost-aware Targeted Viral Marketing - CTVM): Given a graph and its diffusion model $\mathcal{G} = (V, E, c, b, \mathcal{D})$ and a budget $\kappa > 0$, find a seed set $S \subset V$ with the total cost $c(S) \leq \kappa$ to maximize the benefit $\mathbb{B}(S)$.

IV. Multi-Stage Programming Formulations

In this section, we present two variants of EXACT solutions, T-EXACT and E-EXACT and their corresponding sample average approximation T-SAA and E-SAA, respectively. T-EXACT is based on two-stage stochastic programming while E-EXACT is the edge-based formulation with cycle-elimination. We then discuss the scalability issue of traditional stochastic programming which is later verified in our experiment, shown in Section VII.

A. Two-Stage Stochastic Linear Program (T-EXACT)

Given an instance $\mathcal{G} = (V, E, c, b, p)$ of CTVM, we first use integer variables $s_v, v \in V$ to represent whether or not node v is selected as a seed node. That is, $s_v = 1$ if v is selected; otherwise $s_v = 0$.

Variables s are known as first stage variables. The values of s are to be decided before the actual realization of the uncertain parameters in G. Further, the set of selected nodes need to satisfy the budget constraint $\sum_{v \in V} s_v c_v \leq \kappa$.

Given a random graph G, we define variable $x_v, v \in V$, to be the activation state of node v when the propagation stops. That is, $x_v = 1$ if v is eventually activated; otherwise $x_v = 0$.

The benefit obtained by seed set can be computed using a second stage mixed integer programming, denoted by $B(s, x, G)$ as follows.

$$B(s, x, G) = \max \sum_{v \in V} b(v) x_v \tag{4}$$

$$\text{s. t.} \sum_{u \in IR(G,v)} s_u \geq x_v, \ v \in V, \tag{5}$$

$$s_u \in \{0, 1\}, x_v \in [0, 1] \tag{6}$$

where $IR(G, v)$ denotes the set of nodes u so that there exists a path from u to v in G.

The two-stage stochastic linear formulation for the CTVM problem is as follows.

$$\max_{s \in \{0,1\}^n} \quad \mathbb{E}\left[B(s, x, G)\right] \tag{7}$$

$$\text{s. t.} \sum_{v \in V} s_v c_v \leq \kappa \tag{8}$$

$$\text{where } B(s, x, G) \text{ is given in (4)-(6)} \tag{9}$$

The objective is to maximize the expected benefit of the activated nodes $\mathbb{E}\left[B(s, x, G)\right]$, where $B(s, x, G)$ is the optimal value of the second-stage problem. This stochastic programming problem is, however, not yet ready to be solved with a linear algebra solver.

1) Discretization: To solve a two-stage stochastic problem, one often needs to discretize the problem into a single (very large) linear programming problem. That is we need to consider all possible realizations $g \in \Omega$ and their probability masses $\mathsf{Pr}[G = g]$. The two-stage stochastic program can be discretized into a mixed integer programming, denoted by MIP$_F$ as follows.

$$\max \sum_{g \in \Omega} \mathsf{Pr}[G = g] \sum_{v} b(v) x_v^l \tag{10}$$

$$\text{s. t.} \sum_{v \in V} s_v c_v \leq \kappa \tag{11}$$

$$\sum_{u \in IR(g,v)} s_u \geq x_v^g, \ v \in V, g \in \Omega \tag{12}$$

$$s_u \in \{0, 1\}, x_v^g \in [0, 1] \tag{13}$$

B. Sample Average Approximation T-SAA

An approach to reduce the number of realizations in T-EXACT is to apply the Sample Average Approximation (SAA) method. In that method, we generate independently T graph samples $G^1, G^2, \cdots, G^T$ from $\mathcal{D}$.

The expectation objective $q(s) = \mathbb{E}[B(s, x, G)]$ is then approximated by the sample average $\hat{q}_T(x) = \frac{1}{T} \sum_{l=1}^{T} \sum_v (b(v) x_v^l)$, and the new formulation is then

$$\max \frac{1}{T} \sum_{l=1}^{T} \sum_{v} b(v) x_v^l$$
$$\text{s. t. Constraints } (11) - (13), \tag{14}$$

where $x_v^l \in [0, 1]$ is an abbreviation for $x_v^{G^l}$.

We shall refer to the above mixed integer linear programming as T-SAA.

We start with identifying the number of samples T needed to guarantee an ϵ error, followed by the expected (exponential) time complexity to solve the above mixed integer linear programming with T samples.

1) Sample Complexity: We bound the concentration with Hoeffding's inequality

Lemma 1 (Hoeffding's Inequality): Let $X_1, \ldots, X_n$ be independent random variables in $[0,1]$. Let $\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$. Then, we have $\Pr[|\bar{X} - X| > t] \leq 2\exp\left(-2nt^2\right)$.

Let $\mathcal{C} = \{S \subseteq V : \sum_{v \in S} s_v c_v \leq \kappa\}$ be the set of all candidate seed sets. In the worst-case, $\mathcal{C}$ can contains exponentially many candidates.

The following lemma gives a bound on the number of necessary samples to guarantee an $\epsilon\sigma_B$ *additive error.*

Lemma 2: For fixed $T = \Omega(\frac{1}{\epsilon^2}\log\frac{2}{\delta}\log|\mathcal{C}|)$, we have

$$\Pr[|\bar{B}(S) - \mathbb{B}(S)| > \epsilon\sigma_B] \leq \delta,$$

Proof: For a fixed candidate solution $S \in \mathcal{C}$, apply the Hoeffding's inequality on $\frac{1}{\sigma_B}B(G^1, S), \frac{1}{\sigma_B}B(G^2, S), \ldots, \frac{1}{\sigma_B}B(G^T, S)$, we have

$$\Pr[|\bar{B}(S) - \mathbb{B}(S)| > \epsilon\sigma_B] \leq 2\exp\left(-2\epsilon^2 T\right).$$

By choosing $T = \Omega\left(\frac{1}{\epsilon^2}\log\frac{2}{\delta}\log|\mathcal{C}|\right)$ and taking the union bound over all candidate solutions in $\mathcal{C}$, we obtain the desired error bound. ■

In the worst-case, $\mathcal{C} = 2^V$, the powerset of V, thus, $|\mathcal{C}| = 2^n$ and $T = O(n1/\epsilon^2)$.

The estimation on T maybe too conservative for practical estimates. While it provides some evidence on the convergence of the solution, it is excessively large for practical purposes.

2) Time Complexity: We analyze the time complexity of T-SAA in Eq. 14, assuming an exhaustive search on $0-1$ integer variables $s_v, v \in V$. While the branch-and-cut (and other mixed integer linear programming methods) performs much better in practice, it has the same worst-case time complexity.

For each of the 2^n possible assignments of s_v, we need to solve a remaining linear programming of nT random variables x_v^l of size, measured by the number of non-zeros, M. The expectation of M depends on T and the expected influence of nodes in G as characterized in the following lemma.

Lemma 3: The expected size, measured by the number of non-zeros, of T-SAA is $\mathbb{E}[M] = T \times \sum_{u \in V} I(u)$, where $I(u)$ denotes the expected influence of node $u \in V$.

Proof: For each $u \in V$, the number of constraints (12) that u was on the left hand sides equal the number of nodes that u can reach to. Thus, the expected number of constraints (12) that u participates into is, hence, $I(u)$. Taking the sum over all possible $u \in V$, we have the expected size of the T-EXACT with T graph realizations is $T \times \sum_{u \in V} I(u)$. ■

In other words, for each G^l, the size of T-SAA increases by the size of transitive closure in G^l, i.e., the number of pairs (u, v) that u can reach to v. For many subgraphs, this increase is of $O(n^2)$, making T-SAA bloats rapidly as T increases.

To bound the time complexity of solving the LP, we use the following time bound on Karmarkar's algorithm

Theorem 1 [25]: Denoting by N the number of variables and L the number of bits of input in a linear programming, then the runtime of Karmarkar's algorithm is

$$O(N^{3.5}L^2 \cdot \log L \cdot \log\log L).$$

Substitute $N = nT = O(\frac{1}{\epsilon^2}n^2)$ and $L = O(M) = O(T \times \sum_{u \in V} I(u)) = O(\frac{1}{\epsilon^2}n^3)$ from Lemma 3, we obtain the approximate time to solve T-SAA as

$$O\left(2^n n^7 1/\epsilon^7 1/\epsilon^4 n^6 \ polylog(n)\right)$$

Simplify and we get the following result

Lemma 4: The worst-case time complexity of T-SAA is $O\left(2^n n^{13}\frac{1}{\epsilon^{11}} polylog(n)\right)$.

C. Edge-Based Formulation With Cycle-Elimination (E-EXACT)

We provide an alternative formula in which for each random graph G, the size of the mixed integer linear program increases by $O(|E|)$ rather than $O(n^2)$ as in T-EXACT.

It is important that we formulate the second stage as a maximum problem, so that we will obtain a bi-level $\mathbf{Max} - \mathbf{max}$ optimization. The alternative $Max - min$ formulation is not only more sophisticated but also constrained to small size instances in practice.

The main idea is *to build a cascade tree from the seed nodes and count the number of activated nodes* instead of counting the number of activated nodes like in T-EXACT. Given a random graph $G = (V, E)$, for each $(u, v) \in E$ we define a variable $y_{uv} = \begin{cases} 1 \text{ if the edge } (u,v) \text{ is "active"}, \\ 0 \text{ otherwise.} \end{cases}$

To "build" a cascade tree, we constraint that each node v in V has at most one active edge (u, v) going to v, as shown in Eq. (17). In addition, (u, v) is active if and only if a) (u, v) is an edge on the graph realization and b) either u is selected, i.e., $s_u = 1$, or there exists active edge (w, u) going to u, see Eq. (16). Moreover, we forbid cycles composed of all active edges. This is similar to the sub-tour elimination for the TSP problem [26]. There might be an exponential number of cycles, however, the cycle can be added gradually. In each step, we identify a cycle of which constraint is violated and add the constraint to the programming formulation. Given a fractional solution $(s; y)$, an exact separation algorithm for some class of inequalities either finds a member of the class violated by $(s; y)$ or proves that no such member exists. There is an exact algorithm for the separation procedure based on finding the shortest path as follow.

Let $z_{uv} = 1 - y_{uv}$, the constraint (18) can be rewritten as $\sum_{(u,v) \in C} z_{uv} \geq 1$ Thus we can find violated constraint by looking for the smallest length cycles in the graph with edges' lengths z_{uv}. In that graph, each edge (u, v) with $z_{u,v} + d(u, v) < 1$, where $d(u, v)$ denotes the shortest distance between u and v, will correspond to an violated constraint. The major time complexity in finding the violated cycles is on finding all-pair-shortest paths which can be solved in $O(n^2 \log n + nm)$ using the Johnson's algorithm.

Since each activated node u is either already in the seed set or activated by exactly one neighbor in the built cascade tree, the objective and the complete formulation is then as follows.

$$B_E(s, x, G) = \max \sum_{(u,v)\in E} b(v) y_{uv} + \sum_{u \in V} b(u) s_u \quad (15)$$

$$\text{s. t.} \sum_{w \in N^-(u)} y_{wu} + s_u \geq y_{uv}, \quad (u,v) \in E \quad (16)$$

$$\sum_{u \in N^-(v)} y_{uv} + s_v \leq 1, \quad v \in V \quad (17)$$

$$\sum_{(u,v)\in C} y_{uv} \leq |C| - 1, \quad \text{any cycle } C \quad (18)$$

$$s_u \in \{0, 1\}, \; u \in V, \quad (19)$$

$$y_{uv} \in [0, 1], \quad (u,v) \in E \quad (20)$$

The two-stage stochastic linear formulation for the CTVM problem is as follows.

$$\max_{s \in \{0,1\}^n} \quad \mathbb{E}\left[B_E(s, x, G)\right] \quad (21)$$

$$\text{s. t.} \sum_{v \in V} s_v c_v \leq \kappa \quad (22)$$

$$\text{where } B_E(s, x, G) \text{ is given in (15)-(20)} \quad (23)$$

The following lemma proves the one-to-one mapping between activated nodes (not in the seed) and the active edges in the cascade tree.

Lemma 5: The number of activated nodes will be the sum of the number of active edges plus the number of seed nodes.

Proof: Define $A_E = \{(u, v)|y_{uv} = 1\}$ the set of active edges and $A_V = S \cup T$, where $S = \{u|s_u = 1\}$ and $T = \{v \text{ reachable from some } u \in S \text{ via a path of only active edges}\}$. We need to show that $|A_V| = |S| + |A_E|$ or equivalently we need to show

$$|A_E| = |T|.$$

The constraints (19) guarantee that each node $v \in V$ has at most one incoming active edge. The constraints (20) forbid cycles to form among the active edges, thus each active edges will point to exactly one active node that is not in S (otherwise we would have a cycle). Thus, we have an one-to-one mapping between the active edges and the active nodes that are not in S, i.e., $|A_E| = |T|$. ∎

D. Sample Average Approximation E-SAA

Similarly, we construct a Sample Average Approximation, called E-SAA. Given T graph samples, $G^1, G^2, \cdots, G^T$ from $\mathcal{D}$, we have

$$\max \frac{1}{T} \sum_{l=1}^{T} \sum_{(u,v)\in E} b(v) y^l_{uv} + \sum_{u \in V} b(u) s_u \quad (24)$$

$$\text{s. t.} \sum_{v \in V} s_v c_v \leq \kappa \quad (25)$$

$$\text{Constraints (16)-(20) for } G^l, l = 1..T \quad (26)$$

We shall refer to the above mixed integer linear programming as E-SAA.

1) Sample Complexity and Time Complexity of E-SAA: Similar to that in Lemma 2, we can obtain the same bound on the number of samples.

Lemma 6: For fixed $T = \Omega(\frac{1}{\epsilon^2} \log \frac{2}{\delta} \log |\mathcal{C}|)$*, we have*

$$\Pr[|\bar{B}_E(S) - \mathbb{B}(S)| > \epsilon \sigma_B] \leq \delta,$$

Since the number of sub-tour elimination constraints in Eq. (18) can be exponentially many, in theory the worst-case complexity of E-SAA is much worse than that in T-SAA. However, in practice, those constraints are added gradually and potentially lead to a more overall efficient formulation.

E. Scalability Issues of Traditional Stochastic Optimization

While both T-EXACT and E-EXACT (called EXACT for short) are designed based on a standard method for stochastic programming, traditional methods can only be applied for small networks, up to few hundreds nodes [12].

There are three major scalability issues when applying SAA and using EXACT for the influence maximization problem. First, the samples have a large size $O(m)$. For large networks, m could be of size million or billion. As a consequence, we can only have a small number of samples, sacrificing the solution quality. For billion scale networks, even one sample will let to an extremely large ILP, that exceeds the capability of the best solvers. Second, the theory developed to assess the solution quality such as those in [12] and [13], only provide approximate confidence interval. That is the quality assessment is only justified for sufficiently large samples and may not hold for small sample sizes. And third, most existing solution quality assessment methods [12], [13] only provide the assessment for a given number of sample size. Thus, if the quality requirement is given a priori, e.g., (ϵ, δ) approximation, there is not an efficient algorithmic framework to identify the number of necessary samples.

V. TipTop - An Efficient $(1 - \epsilon)$-Optimal Solution

In this section, we introduce our main contribution TipTop, which is the first algorithm that can return a $(1 - \epsilon)$-approximation ratio w.h.p of $(1 - \delta)$ where δ is given a priori. It overcomes the above mentioned scalability issues and can run on billion-scale networks.

A. TipTop *Algorithm Overview*

For readability, we start with the solution to CTVM in which all nodes have uniform cost. Let $\kappa = k$, we want to find S with $|S| \leq k$ so as to maximize the benefit $\mathbb{B}(S)$. Note that benefit function is still heterogeneous. The solution to non-uniform cost is presented later in Subsection VI-B.

At a high level, TipTop first generates a collection $\mathcal{R}$ of random hyperedges sets which serves as the searching space to find a candidate solution $\hat{S}_k$ to CTVM. It next calls the Verify procedure which independently generates another collection of random hyperedges sets to closely estimate the objective function's value of the candidate solution. If this value is not close enough to the optimal solution, TipTop generates more samples by calling the IncreaseSamples procedure to enlarge

Algorithm 1 TipTop Algorithm

```
Input: Graph $G = (V, E, b, c, w)$, seed set size $k > 0$, and $\epsilon, \delta \in (0, 1)$.
Output: Seed set $S_k$.
1:  $\Lambda \leftarrow (1+\epsilon)(2+\frac{2}{3}\epsilon)\frac{1}{\epsilon^2}\ln\frac{2}{\delta}$
2:  $t \leftarrow 1; t_{max} = \lceil 2\ln n/\epsilon \rceil; v_{max} \leftarrow 6$
3:  $\Lambda_{max} \leftarrow (1+\epsilon)(2+\frac{2}{3}\epsilon)\frac{2}{\epsilon^2}(\ln\frac{2}{\delta/4} + \ln\binom{n}{k})$
4:  Generate random $\Lambda$ hyperedges sets $R_1, R_2, \ldots$ using BSA [8]
5:  repeat
6:      $N_t \leftarrow \Lambda \times e^{\epsilon t}; \mathcal{R}_t \leftarrow \{R_1, R_2, \ldots, R_{N_t}\}$
7:      $\hat{S}_k \leftarrow$ ILP$_{MC}(\mathcal{R}_t, c, k)$
8:      $< passed, \epsilon_1 > \leftarrow$Verify$(\hat{S}_k, v_{max}, \epsilon, t_{max}, 2^{v_{max}} N_t)$
9:      if (not $passed$) and $(Cov_{\mathcal{R}}(\hat{S}_k) \leq \Lambda_{max})$ then
            $t \leftarrow$ IncreaseSamples$(t, \epsilon, \epsilon_1)$
10: until $passed$ or $Cov_{\mathcal{R}}(\hat{S}_k) > \Lambda_{max}$
11: return $\hat{S}_k$
```

the search space, and thus finding another better candidate solution. When the objective function's value of the candidate solution is close enough to the optimal one, TipTop halts and returns the found solution. The pseudo-code of TipTop is presented in Alg. 1.

As CTVM considers arbitrary benefits, we utilize Benefit Sampling Algorithm – BSA in [8] to embed the benefit of each node into consideration, shown in line 4 of Algorithm 1. BSA performs a reversed influence sampling (RIS) in which *the probability of a node chosen as the source is proportional to its benefit*. Random hyperedges generated via BSA can capture the "benefit landscape". That is they can be used to estimate the benefit of any seed set S as stated in the following lemma.

Lemma 7 [8]: Given a fixed seed set $S \subseteq V$, and let $R_1, R_2, \ldots, R_j, \ldots$ be random hyperedges sets generated using Benefit Sampling Algorithm [8], define random variables

$$Z_j = \begin{cases} 1 & \text{if } R_j \cap S \neq \emptyset, \\ 0 & \text{otherwise.} \end{cases} \tag{27}$$

then

$$\mathbb{E}[Z_j] = \Pr[R_j \cap S \neq \emptyset] = \frac{\mathbb{B}(S)}{\Gamma} \tag{28}$$

where $\Gamma = \sum_{v \in V} b(v)$ is the total nodes' benefit.

For a collection of T random hyperedges sets $\mathcal{R} = \{R_1, R_2, \ldots, R_T\}$, we denote by

$$Cov_{\mathcal{R}}(S) = \sum_{j=1}^{T} Z_j,$$

the number of hyperedges sets that intersect S, and

$$\mathbb{B}_{\mathcal{R}}(S) = \frac{Cov_{\mathcal{R}}(S)}{T} \times \Gamma,$$

the estimation of $\mathbb{B}(S)$ via $\mathcal{R}$.

As shown in Theorem 3, TipTop has an approximation of $(1-\epsilon)$ with a probability of at least $(1-\delta)$. The key point to improve the current best ratio of $(1 - 1/e - \epsilon)$ (and $(1 - 1/\sqrt{e} - \epsilon)$) to $(1-\epsilon)$ lies in solving the Maximum Coverage (MC) problem after generating $\mathcal{R}$ random hyperedges sets of samples (line 7 of Alg. 1). Instead of using greedy technique as in all existing algorithms, we solve the MC *exactly* using Integer Linear Program (ILP) (detailed in subsection V-B).

As exactly solving ILP for MC is NP-hard itself, we need to reduce the time and memory complexities as much as possible. This becomes the solely drive force for the design of our algorithm, which is handled as follows.

First, we need to keep the ILP search space small at the first phase during the searching for the candidate solution $\hat{S}_k$. Else, the ILP solver cannot be executed. It is worth noting that existing solutions only focused on reducing the *total* number of samples generated, not at the searching phase. Relevant to our approach, SSA [7] does have the first phase, however, it has fixed parameter setting, thus cannot achieve optimal number of samples for this phase. This enforces us to carefully generate the first set $\mathcal{R}$ with size Λ as shown in line 1 of Alg. 1. And once $\hat{S}_k$ is not good enough, IncreaseSamples will generate an additional set of samples, which should be dynamically determined in order to meet the requirement of ILP, that is, it should be just large enough to find a near-optimal candidate solution to CTVM. We will discuss more details about IncreaseSamples later in subsection V-C.

Second, in an effort to keep the ILP size small, we need to avoid executing IncreaseSamples as much as possible. Therefore, we need to put more effort in proving the quality of candidate solutions, which is handled by Verify, described in subsection V-C.

We discuss the rest of TipTop in the following subsections.

B. Mixed Integer Linear Programming ILP_{MC}

Given a collection of hyperedges sets $\mathcal{R}$ and the cost $c(v)$ of selecting nodes $v \in V$ and a seek size k, we formulate the following ILP$_{MC}(\mathcal{R}, c, k)$, to find the optimal solution over the generated hyperedges sets $\mathcal{R}$ to the MC problem.

$$\text{maximize} \sum_{R_j \in \mathcal{R}} (1 - y_j) \tag{29}$$

$$\text{subject to} \sum_{v \in V} s_v \leq k \tag{30}$$

$$\sum_{v \in R_j} s_v + y_j \geq 1 \quad \forall R_j \in \mathcal{R} \tag{31}$$

$$s_i \in \{0, 1\} \quad y_j \in [0, 1] \tag{32}$$

Here, $s_v = 1$ iff node v is selected into the seed set, and $s_v = 0$, otherwise. The variable $y_j = 1$ indicates that the hyperedges sets R_j cannot be covered by the seed set ($S = \{v | s_v = 1\}$) and $y_j = 0$, otherwise. The objective aims to cover as many hyperedges sets as possible while keeping the cost at most k using the constraint (30).

We note that the benefit in selecting the node $b(u)$ does not appear in the above ILP as it is embedded in the Benefit Sampling Algorithm (BSA) in [8].

On one hand, the above ILP can be seen as a Sample Average Approximation (SAA) of the CTVM problem as it attempts to find optimal solutions over the randomly generated samples. On the other hand, it is different from the traditional SAA discussed in Sec. IV as it does not require the realization

Algorithm 2 Verify

```
Input: Candidate solution $\hat{S}_k$, $v_{max}$, $\epsilon$, $t_{max}$, and $T_{cap}$.
Output: Passed/not passed and $\epsilon_1$.
 1: $\mathcal{R}_{ver} \leftarrow \emptyset, \delta_2 = \frac{\delta}{4}, cov = 0, \epsilon_1 = \epsilon_2 = \infty$
 2: for $i \leftarrow 0$ to $v_{max} - 1$ do
 3:     $\epsilon_2 = \min\{\epsilon, 1\}/2^i, \epsilon'_2 = \frac{\epsilon_2}{1-\epsilon_2}; \delta'_2 = \delta_2/(v_{max} \times t_{max})$
 4:     $\Lambda_2 = 1 + (2 + 2/3\epsilon'_2)(1 + \epsilon'_2) \ln \frac{2}{\delta'_2} \frac{1}{(\epsilon'_2)^2}$
 5:     while $cov < \Lambda_2$ do
 6:         Generate $R_j$ with BSA [8] and add it to $\mathcal{R}_{ver}$
 8:         if $R_j \cap S \neq \emptyset$ then $cov = cov + 1$
 9:         if $|\mathcal{R}_{ver}| > T_{cap}$ then return $< false, \epsilon_1 >$
10:     end while
11:     $\mathbb{B}_{ver}(\hat{S}_k) \leftarrow \Gamma \frac{cov}{|\mathcal{R}_{ver}|}$, $\epsilon_1 \leftarrow 1 - \frac{\mathbb{B}_{ver}(\hat{S}_k)}{\mathbb{B}_{\mathcal{R}}(\hat{S}_k)}$
12:     if $(\epsilon_1 > \epsilon)$ then return $< false, \epsilon_1 >$
13:     $\epsilon_3 \leftarrow \sqrt{\frac{3 \ln \ (t_{max}/\delta_1)}{(1-\epsilon_1)(1-\epsilon_2) Cov_{\mathcal{R}_t}(\hat{S}_k)}}$.
14:     if $(1 - \epsilon_1)(1 - \epsilon_2)(1 - \epsilon_3) > (1 - \epsilon)$ then
            return $< true, \epsilon_1 >$
15: end for
16: return $< false, \epsilon_1 >$
```

of all random variables, i.e., the status of the edges. Instead, only a local portion of the graph surrounding the sources of hyperedges sets need to be revealed. This critical difference from traditional SAA significantly reduces the size of each sample, effectively, results in a much more compact ILP.

C. The Verify and IncreaseSamples Procedures

As shown in Alg. 2, Verify takes a candidate solution $\hat{S}_k$, precision limit v_{max}, and the maximum number of hyperedges sets T_{cap} as the input. It keeps generating hyperedges sets to estimate $\mathbb{B}(\hat{S}_k)$ until either the relative error reaches $\epsilon/2^{v_{max}-1}$ or the maximum number of generated samples T_{cap} is reached.

Verify uses the stopping rule algorithm in [7] to estimate the influence. It generates a new pool of random hyperedges sets, denoted by $\mathcal{R}_{ver}$. For each pair ϵ'_2, δ'_2, derived from ϵ_2 and δ_2, the stopping rule algorithm will stop when either the cap T_{max} is reached or there is enough evidence (i.e. $cov \geq \Lambda_2$) to conclude that

$$Pr[(1 - \epsilon'_2)\mathbb{B}(\hat{S}_k) \leq \mathbb{B}_{\mathcal{R}_{ver}}(\hat{S}_k) \leq \ (1 + \epsilon'_2)\mathbb{B}(\hat{S}_k)] \geq 1 - \delta'_2.$$

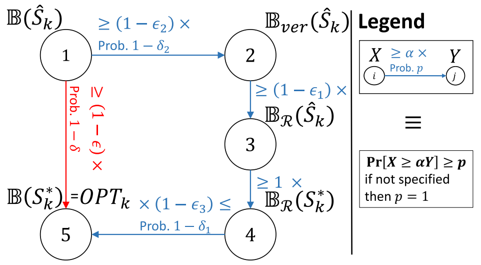
The value of δ'_2 is selected as in line 3 so that the probability of the union of all the bad events is bounded by δ_2.

If the stopping rule algorithm stops within T_{cap} hyperedges sets, the algorithm evaluates the relative difference ϵ_1 between the estimations of $\mathbb{B}(\hat{S}_k)$ via $\mathcal{R}_t$ and $\mathcal{R}_{ver}$. It also estimates the relative gap ϵ_3 between $\mathbb{B}(S^*_k)$ and its estimation using $\mathcal{R}_t$. If the combined gap $(1-\epsilon_1)(1-\epsilon_2)(1-\epsilon_3) < (1-\epsilon)$, Verify returns 'true' and goes back to TIPTOP. In turn, TIPTOP will return $\hat{S}_k$ as the solution and terminate.

If $\hat{S}_k$ does not pass the check in Verify, TIPTOP uses the sub-procedure IncreaseSamples (Alg. 3) to increase the size of the hyperedges sets. Having more samples will likely lead to better candidate solution $\hat{S}_k$, however, also increase the

Algorithm 3 IncreaseSamples

```
Input: $t$ and $\epsilon_1$.
Output: $t$.
1: $\Delta t_{max} = \lceil 2/\epsilon \rceil$
2: return $t + \min\{\max\{\lceil 1/\epsilon \ln \frac{\epsilon_1^2}{\epsilon^2} \rceil, 1\}, \Delta t_{max}\}$
```



$\mathbb{B}(\hat{S}_k)$
$\geq (1-\epsilon_2) \times$
Prob. $1-\delta_2$
$\mathbb{B}_{ver}(\hat{S}_k)$
Legend
1
2
$\geq (1-\epsilon_1) \times$
$\mathbb{B}_{\mathcal{R}}(\hat{S}_k)$
3
Prob. $1-\delta$
$\geq (1-\epsilon) \times$
$\geq 1 \times$
$\mathbb{B}(S^*_k)$=OPT_k
$\times (1-\epsilon_3) \leq$
$\mathbb{B}_{\mathcal{R}}(S^*_k)$
Prob. $1-\delta_1$
5
4
X $\geq \alpha \times$ Y
Prob. p
i j
$\equiv$
$\mathbf{Pr}[X \geq \alpha Y] \geq p$
if not specified then $p = 1$

Fig. 1. Proof map of the main Theorem 3.

ILP solving time. Instead of doubling the current set $\mathcal{R}$ as SSA does, we carefully use the information in the values of ϵ_1 from the previous round together with ϵ to determine the increase in the sample sizes. Recall that the sample size is $e^{t\epsilon}\Lambda$ for increasing integer t. Thus, we increase the sample size via increasing t by (approximately) $\log_{e^\epsilon} \frac{\epsilon_1^2}{\epsilon^2}$. We force t to increase by at least one and at most $\Delta t_{max} = \lceil 2/\epsilon \rceil$. That is the number of samples will increase by a multiplicative factor between $e^\epsilon \approx \ (1 + \epsilon)$ and $e^{\Delta t_{max}} \approx e^2$.

VI. OPTIMALITY OF TIPTOP

In this section, we prove that TIPTOP for arbitrary cost CTVM problem returns a solution $\hat{S}$ that is optimal up to a multiplicative error $1 - \epsilon$ with high probability. Fig. 1 shows the proof map of our main Theorem 3. We first prove the optimality of TIPTOP in the case of the uniform cost and then extend it to arbitrary cost in subsection VI-B.

A. Uniform Cost CTVM

Let $R_1, R_2, R_3, \ldots, R_j, \ldots$ be the random hyperedges sets generated in TIPTOP. Given a seed set S, define random variables Z_j as in (27) and $Y_j = Z_j - \mathbb{E}[Z_j]$. Then Y_j satisfies the conditions of a *martingale* [27], i.e., $\mathbb{E}[Y_i|Y_1, Y_2, \ldots, Y_{i-1}] = Y_{i-1}$ and $\mathbb{E}[Y_i] < +\infty$. This martingale view is adopted from [6] to cope with the fact that random hyperedges sets might not be independent due to the stopping condition: the later hyperedges sets are generated only when the previous ones do not satisfy the stopping conditions. We obtain the same results in [6, Corollaries 1 and 2].

Lemma 8 [6]: Given a set of nodes S and random hyperedges sets $\mathcal{R} = \{R_j\}$ generated in TIPTOP, define random variables Z_j as in (27). Let $\mu_Z = \frac{\mathbb{B}(S)}{\Gamma}$ and $\hat{\mu}_Z = \frac{1}{T}\sum_{i=1}^{T} Z_i$ be an estimation of μ_Z, for fixed $T > 0$. For any $0 \leq \epsilon$, the following inequalities hold

$$\Pr[\hat{\mu} \geq (1 + \epsilon)\mu] \leq e^{\frac{-T\mu\epsilon^2}{2+\frac{2}{3}\epsilon}}, \text{ and} \tag{33}$$

$$\Pr[\hat{\mu} \geq (1 + \epsilon)\mu] \leq e^{\frac{-T\mu\epsilon^2}{3}}, \text{ and} \tag{34}$$

$$\Pr[\hat{\mu} \leq (1 - \epsilon)\mu] \leq e^{\frac{-T\mu\epsilon^2}{2}}. \tag{35}$$

A common framework in [5]–[7] is to generate random hyperedges sets and use the greedy algorithm to select k seed nodes that cover most of the generated hyperedges sets. It is shown in Lemma 3 [5] that $(1-1/e-\epsilon)$ approximation algorithm with probability $1-\delta$ is obtained when the number of hyperedges sets reaches a threshold

$$\theta(\epsilon,\delta) = c \times (8+\epsilon)\left(\ln\binom{n}{k} + \ln\frac{2}{\delta}\right)\frac{n}{OPT_k}\frac{1}{\epsilon^2}, \quad (36)$$

for some constant $c > 0$.

The constant c is bounded to be $8+\epsilon$ in [5], brought down to $8(e-2)(1-1/(2e))^2 \approx 3.7$ in [8] using the zero-one estimator in the work of Dagum et al. [28]. And the current best is $c = 2 + 2/3\epsilon$, inducted from [6, Lemma 6].

Note that $\theta(\epsilon,\delta)$ cannot be used to decide how many hyperedges sets we need to generate since θ depends on the unknown value OPT_k, of which computation is #P-hard.

To overcome that hurdle, a simple stopping rule is developed in [8] to check on whether we have sufficient hyperedges sets to guarantee, w.h.p., a $(1-\epsilon)$ approximation. The rule is that we can stop when we can find *any* seed set S_k, of size k, that coverage $Cov_{\mathcal{R}}(S_k)$ exceeds

$$\Lambda_{max} = (1+\epsilon)\theta(\epsilon,\delta/4) \times \frac{OPT_k}{n} \quad (37)$$
$$= (1+\epsilon)(2+\frac{2}{3}\epsilon)\frac{1}{\epsilon^2}(\ln\frac{2}{\delta/4} + \ln\binom{n}{k}) \quad (38)$$

Our algorithm TIPTOP utilizes this stopping condition to guarantee that at most $O(\theta(\epsilon,\delta))$ hyperedges sets are used in the ILP in the worst-case. As a result, the size of the ILP is kept to be almost linear size, assuming $k \ll n$.

Theorem 2: The expected number of non-zeros in the ILP of TIPTOP *is*

$$O\left(\left(\ln\binom{n}{k} + \ln\frac{2}{\delta}\right)\frac{n}{\epsilon^2}\right).$$

Proof: The expected number of hyperedges sets is $O((1+\epsilon)\theta(\epsilon,\delta))$. Moreover, the expected size of each hyperedges sets is upper-bounded by $\frac{OPT_k}{k}$ ([5, Lemma 4 and eq. (7)]). Thus, the size of the ILP which is equal the total sizes of all the hyperedges sets plus n, the size of the cardinality constraint, is at most $O((1+\epsilon)\theta(\epsilon,\delta)OPT_k) = O\left(\left(\ln\binom{n}{k} + \ln\frac{2}{\delta}\right)\frac{n}{\epsilon^2}\right)$ ■

In practice, the number of required samples in our ILP is many times larger than the minimum number of required samples, i.e., the ILP size is much smaller than the worst-case bound in the above theorem, as shown in Section VII.

Lemma 9: Let S_k^* be a seed set of size k with maximum benefit, i.e., $\mathbb{B}(S_k^*) = OPT_k$. Denote by $\mu_k^* = \frac{OPT_k}{\Gamma}, \delta_1 = \delta/4$ and

$$\epsilon_t^* = \sqrt{\frac{3\ln\ (t_{max}/\delta_1)}{N_t\mu_k^*}}.$$

We have:

$$\Pr[\mathbb{B}_{\mathcal{R}_t}(S_k^*) \geq\ (1-\epsilon_t^*)\mathbb{B}(S_k^*)\ \forall \mathbf{t = 1..t_{max}}] \geq 1-\delta_1.$$

Proof: Apply Lem. 8 (Eq. 34) for seed set S_k^*, mean μ_k^*, ϵ_t^* and N_t samples. For each $t \in [1..t_{max}]$, we have

$$\Pr[\mathbb{B}_{\mathcal{R}_t}(S_k^*) < (1-\epsilon_t)\mathbb{B}(S_k^*)] \quad (39)$$
$$= \Pr\left[\frac{\mathbb{B}_{\mathcal{R}_t}(S_k^*)}{\Gamma} < (1-\epsilon_t)\frac{\mathbb{B}(S_k^*)}{\Gamma}\right] \quad (40)$$
$$< e^{-\frac{3N_t\mu_k^*\epsilon_t^{*2}}{3}} = e^{-\ln(\frac{t_{max}}{\delta_1})} < \delta_1/t_{max}. \quad (41)$$

Taking the union bound over all $t \in [1, t_{max}]$ yields the proof. ■

Lemma 10 (①→②): The probability of the bad event that there exists some set of $t \in [1, t_{max}], i \in [0, v_{max}-1]$, and $\epsilon_2 = \epsilon/2^i$ so that the Verify algorithm returns some bad estimation of $\hat{S}_k$ at line 10, i.e.,

$$\mathbb{B}_{ver}(\hat{S}_k) > \frac{1}{(1-\epsilon_2)}\mathbb{B}(\hat{S}_k),$$

is less than δ_2. Here $\mathbb{B}_{ver}(\hat{S}_k) = \Gamma \times \frac{Cov_{\mathcal{R}_{ver}}(\hat{S}_k)}{|\mathcal{R}_{ver}|}$ be an estimation of $\mathbb{B}(\hat{S}_k)$ using random hyperedges sets in $\mathcal{R}_{ver}$.

Proof: After reaching line 10 in Verify, the generated hyperedges sets within Verify, denoted by $\mathcal{R}_{ver}$, will satisfy the condition that

$$cov = Cov_{\mathcal{R}_{ver}}(\hat{S}_k) \geq \Lambda_2 = 1 + (2+\frac{2}{3}\epsilon_2')(1+\epsilon_2')\ln\frac{2}{\delta_2'}\frac{1}{\epsilon_2'^2},$$

where $\epsilon_2' = \frac{\epsilon_2}{1-\epsilon_2}$.

According to the stopping rule theorem[2] in [28], this stopping condition guarantees that

$$\Pr[\mathbb{B}_{ver}(\hat{S}_k) > (1+\epsilon_2')\mathbb{B}(\hat{S}_k)] < \delta_2'.$$

Substitute $\epsilon_2' = \frac{\epsilon_2}{1-\epsilon_2}$ and simplify, we obtain

$$\Pr[\mathbb{B}_{ver}(\hat{S}_k) > \frac{1}{1-\epsilon_2}\mathbb{B}(\hat{S}_k)] < \delta_2'.$$

The number of times Verify invoked the stopping condition to estimate $\hat{S}_k$ is at most $t_{max} \times v_{max}$. Thus, we can use the union bound of all possible bad events to get a lower-bound $\delta_2' \times t_{max} \times v_{max} = \delta_2$ for the probability of existing a bad estimation of $\hat{S}_k$. ■

②→③: This holds due to the definition of ϵ_1 in Verify. Since $\epsilon_1 \leftarrow 1 - \mathbb{B}_{ver}(\hat{S}_k)/\mathbb{B}_{\mathcal{R}_t}(\hat{S}_k)$, it follows that

$$\mathbb{B}_{ver}(\hat{S}_k) = (1-\epsilon_1)\mathbb{B}_{\mathcal{R}_t}(\hat{S}_k).$$

We note that it is possible that $\epsilon_1 < 0$, and the whole proof still goes through even for that case. In the experiments, we, however, do not observe negative values of ϵ_1.

③→④: Since $\hat{S}_k$ is an optimal solution of ILP$_{MC}(\mathcal{R}, c, k)$, it will be the k-size seed set that intersects with the maximum number of hyperedges sets in $\mathcal{R}$. Let S_k^* be an optimal k-size seed set, i.e., the one that results in the maximum expected benefit[3]. Since $|S_k^*| = k$, it follows that

$$\mathbb{B}_{\mathcal{R}_t}(\hat{S}_k) \geq \mathbb{B}_{\mathcal{R}_t}(S_k^*),$$

[2]replacing the constant $4(e-2)$ with $2+2/3\epsilon$ due to the better Chernoff-bound in Lemma 8

[3]If there are multiple optimal solutions, we break the tie by using alphabetical order on nodes' ids.

where $\mathbb{B}_{\mathcal{R}_t}(\hat{S}_k)$ and $\mathbb{B}_{\mathcal{R}_t}(S^*_k)$ denote the number of hyperedges sets in $\mathcal{R}_t$ that intersect with $\hat{S}_k$ and S^*_k, respectively.

Theorem 3 (Main Theorem ①→⑤): Let $\hat{S}_k$ be the solution returned by TIPTOP (Algorithm 1). We have

$$\Pr[\mathbb{B}(\hat{S}_k) \geq (1-\epsilon)OPT_k] \geq 1-\delta \tag{42}$$

Proof: First we use the union bound to bound the probability of the bad events. Then we show if none of the bad events happen, the algorithm will return a solution satisfying $\mathbb{B}(\hat{S}_k) \geq (1-\epsilon)OPT_k$. The bad events and the bounds on their probabilities are

1) $\Pr[\exists t : \mathbb{B}_{\mathcal{R}_t}(S^*_k) < (1-\epsilon^*_t)\mathbb{B}(S^*_k)\ ] < \delta_1$ (Lem. 9)
2) $\Pr[\exists t, i, \epsilon_2 = \frac{\epsilon}{2^i} : \mathbb{B}_{\mathcal{R}_{ver}}(\hat{S}_k) > \frac{1}{1-\epsilon_2}\mathbb{B}(\hat{S}_k)] < \delta_2$ (Lem. 10)
3) $\Pr[(Cov_{\mathcal{R}_t}(\hat{S}_k) > \Lambda_{max})$ and $(\mathbb{B}(\hat{S}_k) < (1-\epsilon)OPT_k)] < \delta/4$
4) $\Pr[(t > t_{max})$ and $(Cov_{\mathcal{R}_t}(\hat{S}_k) \leq \Lambda_{max})] < \delta/4$

The bounds in 1) and 2) come directly from Lems. 9 and 10. The bound in 3) is a direct consequence of the stopping condition algorithm in [8]. The bound in 4) can be shown by noticing that when $t > t_{max}$ then $N_t \gg \theta(\epsilon, \delta)$. Apply the union bound, the probability that none of the above bad events happen is, hence, at most $\delta_1 + \delta_2 + \delta/4 + \delta/4 = \delta$.

Assume that none of the above bad events happen, we show that TIPTOP returns a $(1-\epsilon)$ optimal solution. If TIPTOP stops with $Cov_{\mathcal{R}_t}(\hat{S}_k) > \Lambda_{max}$, it is obvious that $\mathbb{B}(\hat{S}_k) \geq (1-\epsilon)OPT_k$, since the bad event in 3) do not happen. Otherwise, algorithm Verify returns 'true' at line 13 for some $t \in [1, t_{max}]$ and $i \in [0, v_{max})$.

Follow the path ①→②→③→④. No bad event in 2) implies

$$\mathbb{B}(\hat{S}_k) \geq (1-\epsilon_2)\mathbb{B}_{ver}(\hat{S}_k) \geq (1-\epsilon_2)(1-\epsilon_1)\mathbb{B}_{\mathcal{R}_t}(\hat{S}_k) \tag{43}$$
$$\geq (1-\epsilon_2)(1-\epsilon_1)\mathbb{B}_{\mathcal{R}_t}(S^*_k). \tag{44}$$

No bad event in 1) implies $\mathbb{B}_{\mathcal{R}_t}(S^*_k) \geq (1-\epsilon^*_t)\mathbb{B}(S^*_k)$.

Claim [④→⑤]:

$$\mathbb{B}_{\mathcal{R}_t}(S^*_k) \geq (1-\epsilon_3)\mathbb{B}(S^*_k),$$

To show the above inequality, we prove that

$$\epsilon^*_t = \sqrt{\frac{3\ln\ (t_{max}/\delta_1)}{N_t\mu^*_k}} \leq \epsilon_3 = \sqrt{\frac{3\ln\ (t_{max}/\delta_1)}{(1-\epsilon_1)(1-\epsilon_2)Cov_{\mathcal{R}_t}(\hat{S}_k)}}$$
$$\Leftrightarrow N_t\mu^*_k \geq (1-\epsilon_1)(1-\epsilon_2)Cov_{\mathcal{R}_t}(\hat{S}_k)$$
$$\Leftrightarrow N_t\mathbb{B}(S^*_k)/\Gamma \geq (1-\epsilon_1)(1-\epsilon_2)N_t\mathbb{B}_{\mathcal{R}_t}(\hat{S}_k)/\Gamma$$
$$\Leftrightarrow \mathbb{B}(S^*_k) \geq (1-\epsilon_1)(1-\epsilon_2)\mathbb{B}_{\mathcal{R}_t}(\hat{S}_k).$$

The last one holds due to the optimality of S^*_k and Eq. 43.

$$\mathbb{B}(S^*_k) \geq \mathbb{B}(\hat{S}_k) \geq (1-\epsilon_1)(1-\epsilon_2)\mathbb{B}_{\mathcal{R}_t}(\hat{S}_k).$$

Combine Eq. 44 and the above claim, we have

$$\begin{aligned}\mathbb{B}(\hat{S}_k) &\geq (1-\epsilon_2)(1-\epsilon_1)\mathbb{B}_{\mathcal{R}_t}(S^*_k)\\ &\geq (1-\epsilon_2)(1-\epsilon_1)(1-\epsilon_3)\mathbb{B}(S^*_k)\\ &\geq (1-\epsilon)OPT_k.\end{aligned}$$

The last one holds due to the terminating condition $(1-\epsilon_2)(1-\epsilon_1)(1-\epsilon_3) < (1-\epsilon)$ in line 13 in the Verify algorithm. ■

B. Arbitrary Cost CTVM

We now consider the case of heterogeneous cost. Note that TIPTOP and its proofs in subsection VI-A already considered the heterogeneous benefit function $b(.)$ thus we only discuss the arbitrary cost function $c(.)$ in this subsection.

Changes in the Algorithm: With heterogeneous selecting cost, seed sets may have different sizes. We define $k_{max} = \max\{k : \exists S \subset V,\ |S| = k, c(S) \leq \kappa\}$. The value of Λ_{max} at line 3 of Alg. 1 will be defined as follows:

$$\Lambda_{max} = (1+\epsilon)(2+\frac{2}{3}\epsilon)\frac{1}{\epsilon^2}[\ln\ (8/\delta) + \min\{k_{max}\ln n, n\}].$$

Line 7 of Alg. 1 will be replaced by $\hat{S}_k \leftarrow \text{ILP}_{MC}(\mathcal{R}_t, c, \kappa)$ where we pass the value κ instead of k.

In addition, the cardinality constraint (30) will be changed into a knapsack constraint $\sum_{v\in V} c(v)s_v \leq \kappa$, where $c(v)$ is the cost of selecting node v and κ is the given budget.

The Verify and IncreaseSamples procedures are kept intact.

Theorem 4: The generalized TIPTOP *as discussed above has an approximation ratio of* $(1-\epsilon)$ *with high probability.*

Proof: We follow the same proof map as in subsection VI-A. All the previous proofs of the convergence and correctness are still held. Note that we only use Λ_{max} in Lemma 10, in which we apply the inequality $\Lambda_{max} \leq 2n^{2+\epsilon}$. This inequality still holds with the new value of Λ_{max}. ■

1) Time Complexity: Assume that we solve TIPTOP using exhaustive search that provides the same worst-case time complexity as other methods such as branch-and-cut.

The number of variables y_j in TIPTOP equals to the number of hyperedges and is bounded by $O(nk\log n\frac{1}{\epsilon^2\ OPT_k})$ from Eq. (36). Since $OPT_k \geq k$, the number of variables in TIPTOP is $N_{TipTop} = O(n\log n\frac{1}{\epsilon^2})$.

For each of the 2^n possible assignments of s_v, we need to solve a remaining linear programming of size

$$M_{TipTop} = O\left(\left(\ln\binom{n}{k} + \ln\frac{2}{\delta}\right)\frac{n}{\epsilon^2}\right) \tag{45}$$
$$= O\left(nk\ln n\frac{1}{\epsilon^2}\right) = O(n^2\log n1/\epsilon^2). \tag{46}$$

Apply Theorem 1 and note the high concentration of the number of hyperedges and their total size around the above values, we have,

Lemma 11: The expected time to solve TIPTOP *will be*

$$O\left(2^n n^{3.5}\frac{1}{\epsilon^7}n^4\frac{1}{\epsilon^4}polylog(n)\right) = O\left(2^n n^{7.5}\frac{1}{\epsilon^{11}}polylog(n)\right).$$

In theory, the worst-case of TIPTOP (with a multiplicative ϵ error guarantee) is better than T-SAA (with an additive error $\epsilon\sigma_B$ guarantee) by a factor $n^{5.5}$. In practice, TIPTOP is also much more efficient due to its simple constraints.

VII. EXPERIMENTS

We conduct several experiments to illustrate the performance and utility of TIPTOP. First, the performance of TIPTOP is compared to both T-SAA and E-SAA. Our results show that it is magnitudes faster while maintaining solution quality (sec. VII-A). We then apply TIPTOP as a benchmark

TABLE I
NETWORKS USED IN OUR EXPERIMENTS

Dataset	**Network Type**	**Nodes**	**Edges**
US Pol. Books [29]	Recommendation	105	442
GR-QC [30]	Collaboration	5242	14496
Wiki-Vote [30]	Voting	7115	103689
NetPHY [31]	Collaboration	37149	180826
Slashdot [30]	Social	82168	948464
Twitter [32]	Social	41M	1.5B

for existing methods, showing that they perform better than their guarantee in certain cases—but have degraded performance in others. Finally, we conclude the section with an in-depth analysis of TIPTOP's performance, with a focus on its sampling behavior.

We implemented TIPTOP in Rust using Gurobi to solve the IP.[4] Unless noted otherwise, each experiment is run 10 times and the results averaged. Settings for ϵ are listed with each experiment, but δ is fixed at $1/n$. All influence values are obtained by running a separate estimation program with $\epsilon = 0.02$ using the seed sets produced by each algorithm as input. Throughout this section, we scale solution quality by an *upper bound* on the optimal solution. When comparing to T-SAA and E-SAA, we treat the maximal performance of these algorithms as optimal to show the performance of TIPTOP. Subsequently, we assume that TIPTOP exactly matches its approximation guarantee, which places an upper bound on the optimal of $1/(1-\epsilon)$ times the TIPTOP solution.

A. Comparison to the Exact IPs

Since both T-SAA and E-SAA produce exact results for influence maximization, we use them to show the optimality of TIPTOP. As the SAA-based methods lack scalability, we run on the 105-node US Political Books network only. As shown in Fig. 2, TIPTOP consistently performs as well as T-SAA while performing more than ten times faster even despite the threading difference. Note that the decreasing runtime of TIPTOP with increasing k is on the scale of 5-10 seconds and can easily be explained as variation in the number of samples needed by TIPTOP and in running time of the IP solver. Different behaviors are observed in running time for T-SAA and E-SAA between normalized and random costs. The reason is that the sampling procedure in T-SAA and E-SAA do not take the node benefit in the account. In contrast, the sampling algorithm in TIPTOP will sample hyperedges starting from high-benefit nodes more often. Thus, the TIPTOP running time is more stable across cost settings.

B. Benchmarking Greedy Methods

Having established that TIPTOP is capable of producing almost exact solutions significantly faster than other IP-based solutions, we now exploit this property to place an upper bound on the performance of other algorithms. Ultimately, this allows us to make statements about the *absolute* performance of these algorithms rather than merely their *relative*

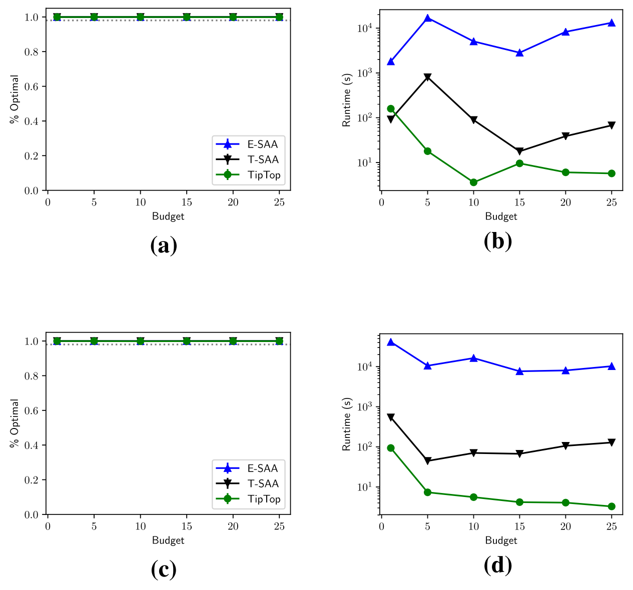


Fig. 2. The performance of TIPTOP ($\epsilon = 0.02, \delta = 1/n$), T-SAA, and E-SAA ($T = 5000$) on the US Political Books [29] network under the cost-aware problem setting. Costs are (a-b) normalized by node degree and (c-d) assigned uniformly at random on $[0, 1)$. The y-axes of Figures (b), (d) are log-scaled. (a) Solution quality (normalized linear costs). (b) Running time (normalized linear costs). (c) Solution quality (random costs). (d) Running time (random costs).

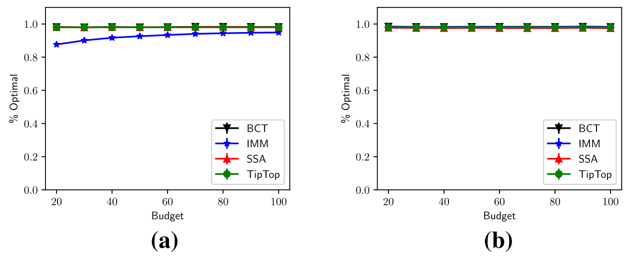


Fig. 3. Mean performance of each approximation algorithm as the budget is varied under the Unweighted and Cost-Aware problem settings with $\epsilon = 0.02$. OPT is estimated assuming that TIPTOP achieves exactly $(1-\epsilon)OPT$. (a) Unweighted NetPHY. (b) Cost-aware NetPHY.

performance. The algorithms we examine are IMM, BCT, and SSA under three problem settings across four networks. All evaluations are under the IC model with edge probabilities set to $p(u, v) = 1/d_{\text{in}}(v)$, where $d_{\text{in}}(v)$ is the in-degree of v. This weight setting is adopted from prior work [6], [7].

We first consider the traditional IM problem, referred to as *Unweighted* here to distinguish it from the subsequent problems. The authors' implementations of each algorithm are applied directly. Fig. 3a shows that the decade-or-so of work on this problem has resulted in greedy solutions that far exceed their guarantee of $1-1/e-\epsilon$. Interestingly, this pattern continues under the *Cost-Aware* setting (Fig. 3b).

Parameter Settings: The Cost-Aware setting generalizes the Unweighted setting by adding a cost to each node on the network. In a social network setting, the costs can be understood as the relative price each user sets for their participation in the marketing campaign. Note that neither IMM nor SSA support costs natively. We extend both to this problem by scaling

[4]Code available at `https://github.com/emallson/tiptop`

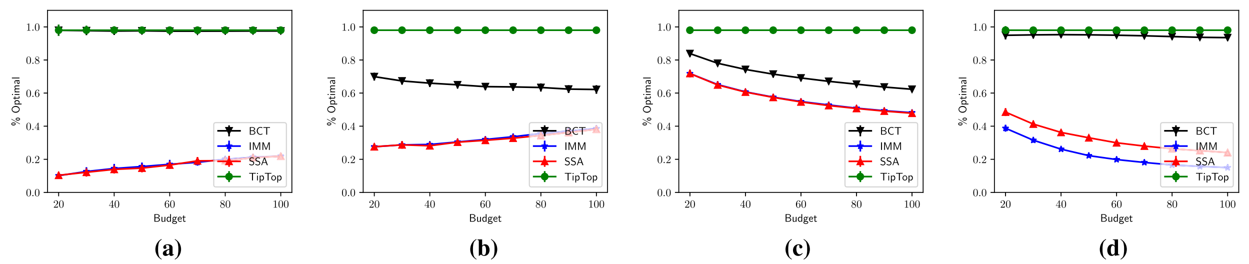


Fig. 4. Mean performance as the budget is varied under the CTVM problem setting with linear costs. Note that IMM and SSA have no guarantees on the CTVM setting. (a) GR-QC. (b) Wiki-Vote. (c) NetPHY. (d) Slashdot.

their objective functions by cost and limiting the number of selected nodes by the sum of costs instead of the number of nodes, which gives each an approximation guarantee of $1-1/\sqrt{e}-\epsilon$ [20]. We consider three ways users may determine their costs.

Random Costs. First, users may determine the value of their influence independently from the network. We model this with Random Costs: each node selects a cost at random on $[0, 1)$.

Degree-Normalized Costs. Users may instead use the most readily-available metric of their relative importance to determine prices: follower count. We examine two different functions of this, **Normalized Linear Costs** and **Normalized Logarithmic Costs**, to develop an understanding of how this impacts algorithm performance. In the linear case, each user sets its cost as $\text{cost}(u) = \frac{n}{|E|} d_{\text{out}}(u)$ ($d_{\text{out}}(u)$ is the out-degree of u). The logarithmic case wraps the degree component: $\text{cost}(u) = \frac{n}{|E|} \log(d_{\text{out}}(u))$. The $n/|E|$ term normalizes costs across networks to allow using the same budget on each. These two cases function as rough upper and lower bounds on reasonable behavior, as shown by the following example.

Suppose two users are setting prices, one with two thousand followers and one with two million. In the linear case if the former user demands \$800 to become an influencer, then the latter user will demand \$800,000. On the other hand, in the log case the latter will demand only \$1,527 – slightly less than double that demanded by the two-thousand-follower user. We find that the performance remains similar to the unweighted case under degree scaling (see Fig. 3).

Benefit Settings: Lastly, we consider the full **CTVM** problem described above. We target 5% of each network at random, assigning each targeted user a *benefit* on $[0.1, 1)$ and each non-targeted user a benefit of 0. Costs are assigned according to one of the previous models (Random, Degree-Normalized Linear or Logarithmic). Neither IMM nor SSA can be extended to this problem without significant modifications, and therefore neither incorporates benefits. Fig. 4 shows the performance on the GR-QC, Wiki-Vote, NetPHY and Slashdot networks.

From Fig. 4, we can see that the topology has a significant impact on the performance of each algorithm. Further, on NetPHY both IMM and SSA do astonishingly well despite being ignorant of the targeted nodes. However, their behavior remains inconsistent across datasets due to their ignorance of the varying benefit assignments. Figure 5 shows that BCT performs similarly regardless of cost function, though random costs can cause notably worsened performance (Fig 5a).

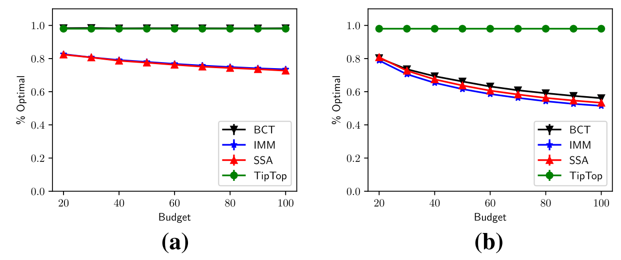


Fig. 5. Performance on the Slashdot network under the CTVM setting with alternate costs. (a) Random. (b) Log-Outdegree.

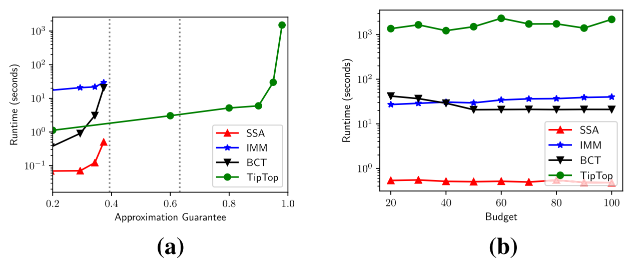


Fig. 6. Mean running time on the NetPHY network under the CTVM problem setting as ϵ and k are varied. (a) Runtime as ϵ is increased. Budget is held constant at 50. Dotted lines show $1-1/\sqrt{e}$ and $1-1/e$, respectively. (b) Runtime as budget is increased. ϵ is held constant at 0.02.

C. Analyzing TipTop's Performance

We now turn our attention to dissecting the performance of TipTop in more detailed. Even it can be seen from the above section (Fig. 4 and 5) that TipTop outperforms existing solutions, we did not examine the relative costs of running TipTop on these or larger networks.

Running Time: From Fig. 6 we can see that the runtime performance of TipTop is near that of the greedy algorithms. With a comparable approximation guarantee to the greedy methods, we can see that TipTop has truly competitive runtime performance. However, as ϵ approaches 0 the runtime rapidly increases. Fig. 6b illuminates one of the key costs of using an ILP: unpredictability. An ILP solver will in many cases find the solution relatively quickly, but in the worst case the complexity remains exponential. Finally, we note that in our tests on the Twitter dataset (Fig. 7), we are able to solve the unweighted setting with a guarantee of 98% in 13 hours and 28 minutes of CPU time – which, as sampling is easily parallelized, takes *under an hour* with 16 threads on our server. Further, for $\epsilon \geq 0.05$ the runtime of TipTop remains

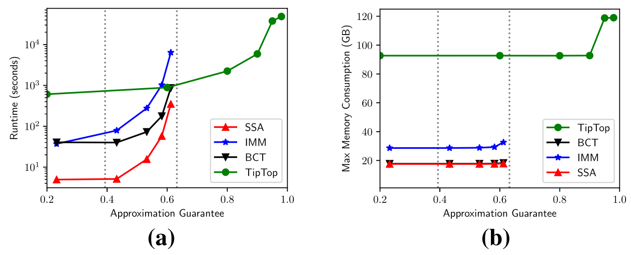


Fig. 7. Running time and memory consumption on the Twitter network under the unweighted problem setting. Only one repetition is used on this dataset. Budget is fixed at 50. Dotted vertical lines show $1 - 1/\sqrt{e}$ and $1 - 1/e$, respectively. (a) Running time. (b) Memory.

TABLE II

Mean Required Samples for Each Algorithm on NetPHY. Approximation Guarantees Are 0.61 for Greedy Methods in the Unweighted Case and 0.37 for the Cost-Aware Case (and the CTVM Case for BCT). The Approximation Guarantee for TipTop Is 0.6 in All Cases. *Coverage*: Samples Input Into the MC Solver. *Verification*: Samples Used to Verify Solution Quality. IMM and BCT Do Not Incorporate a Verification Stage

Coverage	Unweighted	Cost-Aware	CTVM
IMM	3.849×10^7	7.683×10^6	7.683×10^6
SSA	2.361×10^5	5.179×10^5	9.154×10^4
BCT	2.406×10^8	2.261×10^6	1.130×10^6
TipTop	1.215×10^4	6.016×10^4	3.658×10^3
Verification	**Unweighted**	**Cost-Aware**	**CTVM**
SSA	5.383×10^6	1.072×10^6	1.149×10^6
TipTop	3.550×10^5	5.200×10^4	2.900×10^4

competitive with and even outperforms other methods on the same dataset.

Number of Samples: The problem of maximizing influence on billion-scale graphs like Twitter is incredibly difficult – a fact further complicated by the worst-case complexity of solving an IP. TipTop addresses this by dramatically reducing the number of samples needed to solve the problem. Table II shows that on average, TipTop uses 10^3 fewer samples than prior work. Each sample corresponds to one constraint in the IP formulation (specifically, Eqn. 31), and therefore this reduction has a direct impact on the size of the resulting IP.

This is made possible by a SSA-like approach: adding a "Stare" phase to the algorithm. This phase validates the quality of the solution with significantly more samples than the IP uses. Since verification does not require an IP, it has similar complexity to the Stare phase in SSA. The samples used in each algorithm are divided into two sections in Table II. The upper section corresponds to samples used in generating the solution, while the bottom section corresponds to samples used in the Stare phases of SSA and TipTop. Note, however, that TipTop takes a similar number of samples for verification to SSA when obtaining a similar approximation guarantee while also using dramatically fewer for solving the IP.

Memory Consumption: Lastly, we briefly examine the amount of memory consumed by each algorithm. We measure this by running each algorithm with the `time` binary[5] present on Debian, which reports peak memory usage in kilobytes. While for the greedy methods, the number of samples used is the dominating factor, the IP solver used in TipTop may consume an additional large chunk of memory. Figure 7b shows that while this may be true, it does not consume an unreasonable amount of memory. TipTop peaks at 120GB on the Twitter network—a value which we find wholly reasonable for solving with an error of $1 - \epsilon$ on a billion-scale network.

VIII. Conclusion

In this paper, we propose the first (almost) exact and optimal solutions to the CTVM (and thus IM) problem, namely T-EXACT, E-EXACT, and TipTop. T-EXACT and E-EXACT use the traditional stochastic programming approach and thus suffer the scalability issue. In contrast, our TipTop with innovative techniques in reducing the number of samples to meet the requirement of ILP solver is able to run on billion-scale OSNs such as Twitter. TipTop has an approximation ratio of $(1 - \epsilon)$ which significantly improves from the current best ratio $(1 - 1/e - \epsilon)$ for IM and $(1 - 1/\sqrt{e} - \epsilon)$ for CTVM. TipTop also lends a tool to benchmark the absolute performance of existing algorithms on large-scale networks.

References

[1] X. Li, J. D. Smith, T. N. Dinh, and M. T. Thai, "Why approximate when you can get the exact? Optimal targeted viral marketing at scale," in *Proc. INFOCOM*, May 2017, pp. 1–9.
[2] J. Leskovec *et al.*, "Cost-effective outbreak detection in networks," in *Proc. ACM KDD*, 2007, pp. 420–429.
[3] W. Chen, C. Wang, and Y. Wang, "Scalable influence maximization for prevalent viral marketing in large-scale social networks," in *Proc. ACM KDD*, 2010, pp. 1029–1038.
[4] A. Goyal, W. Lu, and L. V. S. Lakshmanan, "SIMPATH: An efficient algorithm for influence maximization under the linear threshold model," in *Proc. ICDM*, Dec. 2011, pp. 211–220.
[5] Y. Tang, X. Xiao, and Y. Shi, "Influence maximization: Near-optimal time complexity meets practical efficiency," in *Proc. ACM SIGMOD*, 2014, pp. 75–86.
[6] Y. Tang, Y. Shi, and X. Xiao, "Influence maximization in near-linear time: A martingale approach," in *Proc. ACM SIGMOD*, 2015, pp. 1539–1554.
[7] H. T. Nguyen, M. T. Thai, and T. N. Dinh, "Stop-and-stare: Optimal sampling algorithms for viral marketing in billion-scale networks," in *Proc. ACM SIGMOD*, 2016, pp. 695–710.
[8] H. T. Nguyen, M. T. Thai, and T. N. Dinh, "Cost-aware targeted viral marketing in billion-scale networks," in *Proc. IEEE INFOCOM*, Apr. 2016, pp. 1–10.
[9] D. Kempe, J. Kleinberg, and É. Tardos, "Maximizing the spread of influence through a social network," in *Proc. KDD*, 2003, pp. 137–146.
[10] C. Borgs, M. Brautbar, J. Chayes, and B. Lucier, "Maximizing social influence in nearly optimal time," in *Proc. 25th Annu. ACM-SIAM Symp. Discrete Algorithms (SODA)*, 2014, pp. 946–957.
[11] H. Nguyen and R. Zheng, "On budgeted influence maximization in social networks," *IEEE J. Sel. Areas Commun.*, vol. 31, no. 6, pp. 1084–1094, Jun. 2013.
[12] A. Shapiro, D. Dentcheva, and A. Ruszczynski, *Lectures on Stochastic Programming: Modeling and Theory*, 2nd ed. Philadelphia, PA, USA: SIAM, 2014.
[13] G. Bayraksan and P. D. Morton, "Assessing solution quality in stochastic programs," *Math. Program.*, vol. 108, no. 2, pp. 495–514, 2006.
[14] U. Feige, "A threshold of ln n for approximating set cover," *J. ACM*, vol. 45, no. 4, pp. 634–652, 1998.
[15] A. Goyal, W. Lu, and L. V. S. Lakshmanan, "Celf++: Optimizing the greedy algorithm for influence maximization in social networks," in *Proc. WWW Companion*, 2011, pp. 47–48.

[5]Note that this is distinct from the bash built-in command.

[16] E. Cohen, D. Delling, T. Pajor, and R. F. Werneck, "Sketch-based influence maximization and computation: Scaling up with guarantees," in *Proc. CIKM*, 2014, pp. 629–638.
[17] T. N. Dinh, Y. Shen, D. T. Nguyen, and M. T. Thai, "On the approximability of positive influence dominating set in social networks," *J. Combinat. Optim.*, vol. 27, no. 3, pp. 487–503, 2014.
[18] N. Ohsaka, T. Akiba, Y. Yoshida, and K.-I. Kawarabayashi, "Fast and accurate influence maximization on large networks with pruned Monte-Carlo simulations," in *Proc. AAAI*, 2014, pp. 138–144.
[19] J. S. He, S. Ji, R. Beyah, and Z. Cai, "Minimum-sized influential node set selection for social networks under the independent cascade model," in *Proc. MobiHoc*, 2014, pp. 93–102.
[20] S. Khuller, A. Moss, and J. S. Naor, "The budgeted maximum coverage problem," *Inf. Process. Lett.*, vol. 70, no. 1, pp. 39–45, 1999.
[21] K. Huang, S. Wang, G. Bevilacqua, X. Xiao, and L. V. Lakshmanan, "Revisiting the stop-and-stare algorithms for influence maximization," *Proc. VLDB Endowment*, vol. 10, no. 9, pp. 913–924, 2017.
[22] H. T. Nguyen, M. T. Thai, and T. N. Dinh, "Stop-and-stare: Optimal sampling algorithms for viral marketing in billion-scale networks," *CoRR*, vol. abs/1605.07990, pp. 1–18, Feb. 2016.
[23] H. T. Nguyen, T. N. Dinh, and M. T. Thai, "Revisiting of 'revisiting the stop-and-stare algorithms for influence maximization,"' in *Proc. Int. Conf. Comput. Social Netw.*, 2018, pp. 273–285.
[24] S. Chen *et al.*, "Online topic-aware influence maximization," *Proc. VLDB Endowment*, vol. 8, no. 6, pp. 666–677, 2015.
[25] N. Karmarkar, "A new polynomial-time algorithm for linear programming," in *Proc. ACM Symp. Theory Comput.*, 1984, pp. 302–311.
[26] G. Dantzig, R. Fulkerson, and S. Johnson, "Solution of a large-scale traveling-salesman problem," *Oper. Res.*, vol. 2, no. 4, pp. 393–410, 1954.
[27] F. Chung and L. Lu, "Concentration inequalities and martingale inequalities: A survey," *Internet Math.*, vol. 3, no. 1, pp. 79–127, 2006.
[28] P. Dagum, R. Karp, M. Luby, and S. Ross, "An optimal algorithm for Monte Carlo estimation," *SIAM J. Comput.*, vol. 29, no. 5, pp. 1484–1496, Mar. 2000.
[29] V. Krebs. *Books About us Politics*. Accessed: Jun. 24, 2017. [Online]. Available: http://www-personal.umich.edu/~mejn/netdata/
[30] J. Leskovec and A. Krevl. (Jun. 2014). *SNAP Datasets: Stanford Large Network Dataset Collection*. [Online]. Available: http://snap.stanford.edu/data
[31] W. Chen, Y. Wang, and S. Yang, "Efficient influence maximization in social networks," in *Proc. KDD*, 2009, pp. 199–208.
[32] H. Kwak, C. Lee, H. Park, and S. Moon, "What is Twitter, a social network or a news media?" in *Proc. WWW*, 2010, pp. 591–600.

Xiang Li (M'18) received the Ph.D. degree from the Computer and Information Science and Engineering Department, University of Florida.

She is currently an Assistant Professor with the Department of Computer Engineering, Santa Clara University. Her research interests include the large-scale optimization and its intersection with cyber-security of networking systems, big data analysis, and cyber physical systems. She has published 25 articles in various prestigious journals and conferences, such as the IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE TRANSACTIONS ON SMART GRIDS, IEEE INFOCOM, and IEEE ICDM, including one Best Paper Award in IEEE MSN 2014, the Best Paper Nominee in IEEE ICDCS 2017, and the Best Paper Award at the IEEE International Symposium on Security and Privacy in Social Networks and Big Data 2018. She has served as the Publicity Co-Chair of the International Conference on Computational Data and Social Networks 2018, the Session Chair of the ACM SIGMETRICS International Workshop 2018, and on the TPC of many conferences, including IEEE ICDCS, IEEE ICDM Workshop, and COCOA, and also served as a reviewer for several journals, such as the IEEE TRANSACTIONS ON MOBILE COMPUTING, the IEEE TRANSACTIONS ON NETWORKS SCIENCE AND ENGINEERING, and the *Journal of Combinatorial Optimization*.

J. David Smith received the B.Sc. degree from the University of Kentucky. He is currently pursuing the Ph.D. degree in computer science with the CISE Department, University of Florida, USA, under the supervision of Dr. M. T. Thai. His current research interests include security on online social networks.

Thang N. Dinh (S'11–M'14) received the Ph.D. degree in computer engineering from the University of Florida in 2013. He is currently an Assistant Professor with the Department of Computer Science, Virginia Commonwealth University. His research interests include security and optimization challenges in complex systems, especially blockchain, social networks, and critical infrastructures. His research won several best papers and best paper nominees from IEEE ICDM, ACM SenSys, and CSoNet. He served as the PC Co-Chair for the BlockSEA'18, COCOON'16, and CSoNet'14 and on the TPC for several conferences, including the IEEE INFOCOM, the ICC, and the GLOBECOM. He is currently an Associate Editor of the *Journal of Computational Social Science* (Springer Nature) and Review Editor of *Big Data*.

My T. Thai (M'06–SM'16) is a UF Research Foundation Professor with the Computer and Information Science and Engineering Department, University of Florida. The results of her work have led to six books and over 150 articles, including the IEEE MSN 2014 Best Paper Award, the 2017 IEEE ICDM Best Papers Award, the 2017 IEEE ICDCS Best Paper Nominee, and the 2018 IEEE/ACM ASONAM Best Paper Runner up. Her current research interests include scalable algorithms, big data analysis, cybersecurity, and optimization in network science and engineering, including communication networks, smart grids, social networks, and their interdependence.

Dr. Thai has been involved in many professional activities. She has been the TPC Chair of many IEEE conferences, and has served as an Associate Editor for the *Journal of Combinatorial Optimization*, *Journal of Discrete Mathematics*, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, and IEEE TRANSACTIONS ON NETWORK SCIENCE AND ENGINEERING and a Series Editor of *SpringerBriefs in Optimization*. She is a the Founding Editor-in-Chief of the *Computational Social Networks Journal*. She has received many research awards, including the UF Provosts Excellence Award for Assistant Professors, the UFRF Professorship Award, the Department of Defense Young Investigator Award, and the National Science Foundation CAREER Award.