Abstract

Imagine we are introducing a new product through a social network, where we know for each user in the network the function of purchase probability with respect to discount. Then, what discount should we offer to those social network users so that, under a predefined budget, the adoption of the product is maximized in expectation? Although influence maximization has been extensively explored, surprisingly, this appealing practical problem still cannot be answered by the existing influence maximization methods. In this paper, we tackle the problem systematically. We formulate the general continuous influence maximization problem, investigate the essential properties, and develop a general coordinate descent algorithmic framework as well as the engineering techniques for practical implementation. Our investigation does not assume any specific influence model and thus is general and principled. At the same time, using the most popularly adopted triggering models as a concrete example, we demonstrate that more efficient methods are feasible under specific influence models. Our extensive empirical study on four benchmark real world networks with synthesized purchase probability curves clearly illustrates that continuous influence maximization can improve influence spread significantly with very moderate extra running time comparing to the classical influence maximization methods.

1 Introduction

Influence maximization \cite{11, 16} is a critical technique in many social network applications, such as viral marketing. The intuition is that, by targeting on only a small number of nodes (called seed nodes), it is possible to trigger a large cascade of information spreading in a social network. Technically, in a social network, influence maximization tries to identify a set of nodes such that if the selected nodes are committed to spread a piece of information to their neighbors, such as adopting a product, the expected spread in the social network is maximized. There have been abundant studies on various models and computational methods for influence maximization. We will review some representative studies in Section 2.
Imagine a company is introducing a new product through a social network by providing discounts to users in the network in the hope of maximizing the influence spread. The total discount is constrained by a budget defined by the company. It is well known that different users in a social network may have a different capability in spreading influence. Consequently, the company naturally wants to offer different users different discounts. It is reasonable to assume that the more discount a user is offered, the more likely the user may adopt the product and spread the influence to her neighbors, which is also known in marketing research as the purchase probability curve being monotonic with respect to discount \([18]\). At the same time, different users may have different purchase probability curves. Given a budget and the users’ purchase probability curves, what discounts should the company offer to the users so that the expected influence spread is maximized? Apparently, this is an interesting question that is asked again and again in various applications where influence maximization is used. At the same time, unfortunately the existing influence maximization techniques cannot answer the question.

Motivated by the practical demands, we investigate the questions about what discounts we should offer to social network users. In general, given a social network, a budget, and, for each user in the network, the seed probability function on discount (corresponding to the purchase probability curve with respect to discount in the above motivation example), the continuous influence maximization problem is to find the optimal configuration, which consists of a discount rate for each user, that maximizes the influence spread in expectation. We make several contributions in this paper.

First, to the best of our knowledge, we are the first to systematically study the problem of continuous influence maximization that utilizes users’ purchase probability curves, which has significant applications in practice. We show that the continuous influence maximization problem is a generalization of influence maximization, which focuses on discrete configurations. Consequently, we investigate the hardness of the problem, and analyze several essential properties of the problem. We do not assume any specific influence model, and thus all properties explored are general.

Second, we develop a general coordinate descent framework for the general continuous influence maximization problem. Again, this algorithm does not assume any specific influence model. Such a coordinate descent algorithmic framework helps us prove some interesting connections of the CIM problem to the traditional influence maximization problem.

Third, we devise practically efficient implementations of our CIM algorithm for specific influence models. We consider triggering models \([16]\), which contain most popularly used influence models like the Independent Cascade model and the linear threshold model in literature. We make an analogy of polling-based influence maximization algorithm \([1]\) and machine learning, and develop effective algorithms also based on polling that avoid the “overfitting” issue.

Last, we report an extensive empirical evaluation using four benchmark real social network data sets with synthesized purchase probability curves. The largest data set has almost 4 million nodes and 70 million edges. The experiment results clearly show that continuous influence maximization can significantly improve influence spread. At the same time, the extra running time remains moderate.

The rest of the paper is organized as follows. We review the related work in Section 2 and formulate the problem of continuous influence maximization in Section 3. In Section 4, we investigate the properties of the expectation of influence spread. We present the general coordinate descent framework and discuss how to find a good initial configuration for running the coordinate descent framework in Section 5. We study in Section 6 the relationship between continuous influence max-
imization developed in this paper and the existing influence maximization problem. In Section 7, we develop algorithms under the triggering models and carefully analyze the “overfitting” issue and how to avoid it. We report an extensive empirical evaluation in Section 8, and conclude the paper in Section 9. To keep the main body of the paper concise, the proofs of all mathematical statements are given in Appendix.

2 Related Work

Domingos et al. [11] proposed to take advantage of peer influence between users in social networks for marketing. The essential idea is that, by targeting on only a small number of users (called seed users), it is possible to trigger a large cascade of users purchasing a specific product through a social network. Consequently, the technical challenge is to find a small set of users who can trigger the largest cascade in the network. Kempe et al. [16] formulated the problem as a discrete optimization problem, which is well known as the influence maximization problem. Since then, influence maximization has drawn much attention from both academia and industry [5, 6, 7, 15, 12, 33, 32, 26].

Most influence maximization algorithms are designed for triggering models [16]. Among these algorithms, a polling-based method [1] has the lowest worst-case time complexity, $O((k + l)(n + m)\log^2 n/\epsilon^3)$. Tang et al. [33, 32] further improved the method to make it run in $O((k + l)(n + m) \log n/\epsilon^2)$ expected time. The empirical studies showed that their improved algorithms are orders of magnitude faster than the other influence maximization algorithms. Lei et al. [19] proposed a method that learns the propagation probabilities while running the viral marketing campaigns. Another line of algorithmic viral marketing research is budgeted influence maximization [35, 27]. Under such problem settings, every user in a social network is associated with a threshold value that indicates the amount of money a company needs to spend to persuade her/him to be an initial adopter. One key problem of this setting is how to obtain users’ threshold values. Singer [29] proposed a mechanism that can elicit users’ true threshold values if they are rational agents. Chen et al. [4] provided a comprehensive survey on influence maximization algorithms.

Farajtabar et al. [14] modeled social events using multivariate Hawkes processes, and developed a convex optimization framework for determining the required level of external incentives, that is, the money spent on users, in order for the network to reach a desired activity level. Although the objective function in [14] is flexible since it only requires that the objective is a concave utility function, both the properties explored in [14] and the algorithm proposed are only suitable for multivariate Hawkes processes rather than a general influence model. Descriptive influence models, such as the independent cascade model and the linear threshold model [16], the two most widely used models, cannot fit in the framework [14].

Eftekhari et al. [13] discovered that sometimes instead of targeting on very few individual users, persuasion attempts on groups of users, for example, displaying advertisements to them, may lead to a wider range of cascades in social networks. The motivation of persuasion on groups is that by spending less money on a targeted individual a company can target at much more users and, as a result, in expectation such a strategy may bring in more initial adopters [34]. Eftekhari et al. [13] assumed that the probability that a user is persuaded to be a seed user is given and fixed, once the user is targeted. A more realistic strategy is that we can adjust the resource spent on a specific individual to manipulate the probability the user becomes a seed user, which is the subject studied in this paper.
Demaine et al. [10] studied the problem of influencing people with partial derivatives (discounts). The output of the method in [10] is similar to ours, which is an $n$-dimensional vector $C \in [0,1]^n$, where $n$ is the number of nodes. But the problem setting of [10] is very different from ours.

First, [10] is based on the Linear Threshold model, while our work does not assume any specific influence model. In this paper, all theoretical results are applicable to submodular and monotonic influence models, and the Linear Threshold model is just one of them. Our implementation in Section 7 can be applied to any triggering models whose reverse propagation can be simulated. Also, the Linear Threshold model is just one of these models.

Second, in [10], a discount $c$ offered to any user $u$ is equivalent to reducing $u$’s threshold $\theta_u$ by $c$, that is, making $\theta_u$ a random variable in $[0, 1 - c]$, no matter what $u$ is. While in reality, how much the discount $c$ affects a user $u$’s threshold should depend on $u$’s purchase probability curve, which is a personalized property of $u$. Thus, [10] actually does not utilize the purchase probability curves of users.

Kempe et al. [17] investigated the general marketing strategies whose problem setting is similar to ours. A major difference in the problem setting is that [17] assumes that all seed probability functions (please refer to Section 3 for the definition) have the “diminishing return” property, which means all seed probability functions are concave or near-concave. In this paper, we do not assume this “diminishing return” property. Instead, a seed probability function can be in any arbitrary shape, even convex. Moreover, the method in [17] is based on discretizing the budget, which means a discount offered to a user is discrete (finitely many), while we provide a totally continuous solution.

As a follow-up work of the preliminary conference version [36] of this paper, Yuan et al. [37] proposed a hill climbing algorithm for the pool setting of our problem. Yuan et al. [37] assumed that there is a discount pool which has finite many discount rates and we can only offer users discounts from the discount pool. This means [37] is similar to [17], because they both discretize the budget. Under such a pool setting, the hill climbing algorithm in [37] is $(1 - \frac{1}{e})$-optimal. However, since the budget is discretized, the hill climbing algorithm [37] does not have the same theoretical guarantees for our problem, which is a fully continuous optimization problem. We will demonstrate in experiments that our methods in [36] still outperform the hill climbing algorithm [37]. Even under the pool setting, where the hill climbing algorithm [37] has a strong theoretical guarantee, our methods in [36], almost every time, can produce better solutions within similar or even shorter running time compared to [37].

Both [17] and [37] did not discuss the complexity of computing the expected influence spread of a given budget allocation plan, but just assumed an oracle to return this value when needed. In our paper, we thoroughly investigate the connection of the complexity of computing the influence spread $I(S)$ and the complexity of computing the expected influence spread $UI(C)$ in section 4.1. Since computing the expected influence spread is often #P-hard (as will be shown in section 4.1) and both [17] and [37] are based on oracles for returning the expected influence spread, implementing methods in [17] and [37] may have some issues. In our paper, we identify the “overfitting” issue caused by the #P-hardness of computing the expected influence spread and provide principled implementations of our algorithms under specific influence models to avoid the “overfitting” issue.

Some results in this paper appeared in a preliminary conference version [36]. This complete version contains some major new results as follows.

First, in [36] we set the number of hyper-edges to a fixed large number when implementing the CIM algorithm. In this paper, we theoretically study the implementation of the CIM algorithms
under triggering models. We firstly propose an algorithm that is guaranteed to converge to a KKT point and has a low computational cost. Then we identify the key problem of implementing CIM algorithms, that is, how many random hyper-edges we need. To solve this problem, we make an analogy between influence maximization and machine learning. By utilizing the idea of uniform convergence in machine learning, we derive the sample size (that is, the number of random hyper-edges) that can avoid the “overfitting” issue in implementing the CIM algorithms. We also borrow the idea of regularization in machine learning to help us reduce the sample size.

Second, the experiments here are conducted in a more principled way compared to our previous work [36]. In [36], we only randomly generated assignments of purchase probability curves of users once, while in this paper, we generate 20 random assignments under each setting and report average results of the 20 different instances for each data set. We also test under a new setting where, instead of a user purchase probability curve, we only have access to the purchase probabilities of users at certain discount rates in a discount pool (for example, only discount rates that are multiples of 1%). We call this setting the pool setting. Results under the pool setting are reported in this paper.

Third, we derive more relations between continuous influence maximization (CIM) and traditional influence maximization (CIM) in Section 6. Corollary 6.3 and Theorem 7 are new. They suggest that we can have a data-dependent approximation of the CIM problem, where the approximation ratio depends on user purchase probability curves.

Last, we provide a more extensive review on related work in Section 2. For example, the Hill Climbing method [37], which is a follow-up work of our preliminary conference version [36], and is \((1 - \frac{1}{e})\)-optimal to the pool setting of our problem. We also compared the Hill Climbing method [37] to our methods in experiments.

3 Problem Definition

A social network is a graph \(G = \langle V, E \rangle\), where \(V\) is a set of users and \(E\) is a set of relationships between users. Denote by \(n = |V|\) the number of users and \(m = |E|\) the number of relationships, that is, edges.

An influence cascading model (influence model for short) describes the process of how influence is cascaded in a social network. Two most widely used influence models are the independent cascade model and the linear threshold model [16]. In an influence cascade process, a cascade is started by a small number of users, whom we call seed users (or seeds for short). We call the set of seed users the seed set, denoted by \(S\). Every influence model has an influence function \(I : 2^V \to R\), where \(I(S)\) is the expected size of the cascade triggered by the seed set \(S\) and is also called the influence spread of \(S\). Usually, \(I(S)\) is assumed monotonic and submodular [16, 24], which capture the intuition about influence spreading.

In this paper, we are interested in customizing a discount for every user in a social network to maximize influence cascading. With a discount of 0%, a user has to pay the full price. With a discount of 100%, the product is free for the user. Please note that the notion of discount here can also be used to model in general the cost that we would like to pay to a user to turn the user into a seed.

Technically, a user \(u \in V\) is associated with a seed probability function \(p_u : [0, 1] \to [0, 1]\), which models the probabilistic distribution that \(u\) is attracted to become a seed user given a discount between 0% to 100%. Denote by \(c_u\) the discount we offer to \(u\). Then, \(p_u(c_u)\) is the probability that \(u\) becomes a seed user given such a discount. In this paper, we assume that a seed probability
function \( p_u(\cdot) \) has the following properties: (1) \( p_u(0) = 0 \); (2) \( p_u(1) = 1 \); (3) \( p_u(c_u) \) is monotonic with respect to \( c_u \); and (4) \( p_u(c_u) \) is continuously differentiable. Conditions (1) and (2) are also assumed in the classical influence maximization problem.

The existing marketing research \([3, 30]\) estimates user purchase probability, where the focus is on the adoption rate of the whole population rather than each individual, and estimations are on specific goods. In reality, a user’s purchasing behavior may change over time and on different types of goods \([8]\). Thus, the best way to decide a user’s seed probability function (purchase probability curve) is to learn from data. Since seed probability functions can take many different forms, it is crucial to design a general marketing method that can handle all kinds of such functions.

We assume that different users become seed users independently. Denote by an \( n \)-dimensional vector \( C = (c_1, ..., c_n) \) a configuration of discounts assigned to all users in \( G \). It is clear that, unlike the situation in the influence maximization problem, the seed set \( S \) in our problem setting is probabilistic. Given a social network \( G = \langle V, E \rangle \) and a configuration \( C \), the probability that a subset \( S \subseteq V \) of users is the seed set is

\[
Pr(S; V, C) = \prod_{u \in S} p_u(c_u) \prod_{v \in V \setminus S} (1 - p_v(c_v))
\]

For a specific influence model with an influence function \( I(S) \), the expected influence spread is

\[
UI(C) = \sum_{S \in 2^V} Pr(S; V, C)I(S)
\]

Now we define the continuous influence maximization problem (CIM for short) studied in this paper as follows. Given a social network \( G = \langle V, E \rangle \), a budget \( B \), a seed probability function \( p_u(c_u) \) for every user \( u \), and an influence model with an influence function \( I(S) \), find the configuration \( C \) that is the optimal solution to the following continuous optimization problem.

\[
\begin{align*}
\text{maximize} & \quad UI(C) \\
\text{s.t.} & \quad 0 \leq c_u \leq 1, \forall u \in V \\
& \quad \sum_{u \in V} c_u \leq B
\end{align*}
\]

We call a configuration satisfying the constraints in Eq. \(3 \) a feasible configuration. Please note that the budget \( B \) here is a safe budget in general. When discounts here are used to model the costs committed to users, the budget models the total cost. When discounts are explained as discount rates, the budget is the worst-case budget. We leave the exploration of other forms of the budget constraint to future work, such as the expected budget under the discount rate explanation.

The classical influence maximization problem is a special case of the continuous influence maximization problem, since it can be written in a similar way as follows.

\[
\begin{align*}
\text{maximize} & \quad UI(C) \\
\text{s.t.} & \quad c_u = 0 \text{ or } c_u = 1, \forall u \in V \\
& \quad \sum_{u \in V} c_u \leq B
\end{align*}
\]

We call a configuration satisfying the constraints in Eq. \(4 \) an integer configuration. Apparently, an integer configuration is also a feasible configuration.
<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G = (V, E)$</td>
<td>Social network $G$, where $V$ is the set of users and $E$ is the set of relationships</td>
</tr>
<tr>
<td>$n =</td>
<td>V</td>
</tr>
<tr>
<td>$m =</td>
<td>E</td>
</tr>
<tr>
<td>$p_u(c_u)$</td>
<td>The probability that $u$ becomes a seed user if $u$ is offered a discount $c_u$</td>
</tr>
<tr>
<td>$C = (c_1, c_2, ..., c_n)$</td>
<td>A configuration, where $c_u$ ($0 \leq c_u \leq 1$) is the discount offered to user $u$</td>
</tr>
<tr>
<td>$Pr(S; V, C)$</td>
<td>The probability of a subset of users $S \subseteq V$ is the seed set $S$ (Eq. 1)</td>
</tr>
<tr>
<td>$UI(C)$</td>
<td>The expected influence spread caused by configuration $C$ (Eq. 2)</td>
</tr>
</tbody>
</table>

Table 1: Frequently used notations.

In the rest of the paper, for the sake of clarity, we also call the classical influence maximization problem discrete influence maximization (DIM for short). Table 1 summarizes the frequently used notations.

4 Expected Influence Spread

The CIM problem is to optimize the expected influence spread $UI(C)$. In this section, we discuss the computation of $UI(C)$ and the monotonicity of $UI(C)$, which prepare us for the solution development in the next sections.

4.1 Computing $UI(C)$

Given $G = (V, E)$, an influence function $I(S)$, and a seed probability function $p_u(c_u)$ for every $u \in V$, how can we obtain $UI(C)$? It is known that, for many popular influence models, computing $I(S)$ is #P-hard [5, 7]. What is the hardness of computing $UI(C)$?

**Theorem 1 (Complexity).** Given a configuration $C$, computing $UI(C) = \sum_{S \subseteq V} Pr(S; V, C)I(S)$ is #P-hard if computing $I(S)$ is #P-hard.

Since $UI(C)$ is the expectation of $I(S)$ over the random variable $S$, we can use Monte Carlo simulations to estimate $UI(C)$. Because every user becomes a seed user independently, randomly generating a seed set $S$ based on $Pr(S; V, C)$ is equivalent to simply adding each user $u$ to $S$ independently with probability $p_u(c_u)$. We have the following result.

**Theorem 2 (($\epsilon, \delta$) estimation).** Suppose we have an influence spread oracle that can return the influence spread $I(S)$ of a given seed set $S$. By calling the influence spread oracle $\frac{n^2 \ln \frac{2}{\delta}}{2^{\rho^2}(\sum_{u \in V} p_u(c_u))^2}$ times, we can have an $(\epsilon, \delta)$ estimation [23] of $UI(C)$.

As mentioned before, computing $I(S)$ is #P-hard for some influence functions. The good news is that there exists a FPRAS [23, 22] for estimating $I(S)$. We prove that if $I(S)$ can be estimated efficiently, so is $UI(C)$. Similar to influence maximization where the number of seeds $B$ is assumed to be $\Omega(1)$, we assume that the expected number of seeds $\sum_{u \in V} p_u(c_u)$ is also $\Omega(1)$. Moreover, as we will analyze in Corollary 6.3 if the expected number of seeds $\sum_{u \in V} p_u(c_u)$ is too small, such a

---

1 An FPRAS (Fully Polynomial Randomized Approximation Scheme) is an algorithm which returns an $(\epsilon, \delta)$ estimation of the desired value in time polynomial to $n$ (size of input), $\frac{1}{\delta}$ and $\ln \frac{1}{\delta}$. 
configuration cannot make a high influence and we are not interested in it if we try to maximize $UI(C)$.

**Theorem 3** (FPRAS estimation). For an influence function $I(\cdot)$, if there is a FPRAS for estimating $I(\cdot)$, there is a FPRAS for estimating $UI(\cdot)$.

For the two most popular influence models, namely the independent cascade model and the linear threshold model [16], computing $I(S)$ has FPRAS [17]. Thus, $UI(C)$ under these two models can be efficiently estimated.

**Corollary 3.1.** Computing $UI(C)$ under both the Independent Cascade model and the Linear Threshold model admits a FPRAS, assuming $\sum_{u \in V} p_u(c_u) \in \Omega(1)$.

In summary, the results in this subsection establish that, as long as $I(S)$ can be computed/estimated efficiently (that is, in polynomial time), $UI(C)$ can also be computed/estimated efficiently. This strong relation makes comparing two different configurations $C_1$ and $C_2$ computationally feasible, since we can efficiently estimate $UI(C_1)$ and $UI(C_2)$ accurately.

4.2 Monotonicity of $UI(C)$

Eq. 3 contains an inequality constraint $\sum_{u \in V} c_u \leq B$. According to the assumption that $p_u(c_u)$ is monotonic with respect to $c_u$, we can show that the inequality constraint $\sum_{u \in V} c_u \leq B$ can be substituted by an equation constraint $\sum_{u \in V} c_u = B$.

**Lemma 1.** Given configurations $C_1 = (c_1^1, ..., c_n^1)$ and $C_2 = (c_1^2, ..., c_n^2)$, if there exists $u$ $(1 \leq u \leq n)$ such that $c_u^1 \geq c_u^2$, and $\forall v \in V \setminus \{u\}$, $c_v^1 = c_v^2$, then $UI(C_1) \geq UI(C_2)$.

For two configurations $C_1 = (c_1^1, ..., c_n^1)$ and $C_2 = (c_1^2, ..., c_n^2)$, we write $C_1 \succeq C_2$ if $\forall u$, $c_u^1 \geq c_u^2$. By the transitivity of $\succeq$ and Lemma 1 we have the following immediately.

**Theorem 4** (Monotonicity). If $C_1 \succeq C_2$, then $UI(C_1) \geq UI(C_2)$.

According to Theorem 4 it is obvious that the optimal $C$ for the continuous influence maximization problem must use up the budget $B$. Thus, the continuous influence maximization problem can be rewritten as follows.

\[
\text{maximize} \quad UI(C) \\
\text{subject to} \quad 0 \leq c_u \leq 1, \forall u \in V \\
\sum_{u \in V} c_u = B
\] (5)

5 A General Coordinate Descent Framework

In this section, we develop a coordinate descent algorithm to solve the continuous influence maximization problem. Our algorithm is general, since we do not compose any constraints on the specific form of the influence function $I(S)$ and the seed probability function $p_u(c_u)$. We only assume that $p_u(c_u)$ is monotonic and continuously differentiable.
5.1 Gradient

For a node $u \in V$, we can rewrite $UI(C)$ as follows.

$$
UI(C) = \sum_{S \in 2^V \cup u \in S} Pr(S; V, C)I(S) + \sum_{S \in 2^V \setminus u \in S} Pr(S; V, C)I(S)
$$

$$
= \sum_{S \in 2^V \setminus \{u\}} Pr(S; V \setminus \{u\}, C)I(S)[1 - p_u(c_u)]
+ \sum_{S \in 2^V \setminus \{u\}} Pr(S; V \setminus \{u\}, C)I(S \cup \{u\})p_u(c_u)
= p_u(c_u) \cdot \sum_{S \in 2^V \setminus \{u\}} Pr(S; V \setminus \{u\}, C)[I(S \cup \{u\}) - I(S)] + const
$$

where $const$ is a constant with respect to $c_u$. Given a graph $G = \langle V, E \rangle$ and influence function $I(S)$, for a node $u \in V$, $Pr(S; V \setminus \{u\}, C)$, $I(S)$ and $I(S \cup \{u\})$ are constants with respect to $c_u$. Therefore, using a node $u \in V$ we can rewrite the objective function $UI(C)$ into a linear function of $p_u(c_u)$, where $c_u$ is the only variable.

Assuming that $p_u(c_u)$ is continuously differentiable, we can take the partial derivative of $UI(C)$ with respect to $c_u$, that is,

$$
\frac{\partial UI(C)}{\partial c_u} = p_u(c_u) \sum_{S \in 2^V \setminus \{u\}} Pr(S; V \setminus \{u\}, C)[I(S \cup \{u\}) - I(S)]
$$

In this way, we can compute the gradient of $UI(C)$ with respect to a specific configuration $C$. The gradient information will be used in the coordinate descent algorithm to be developed next.

5.2 A Coordinate Descent Algorithm

The coordinate descent algorithm is an iterative algorithm. In each iteration, we pick only two variables $c_i$ and $c_j$, and fix the rest $n - 2$ variables. We try to increase the value of the objective function $UI(C)$ by changing only the values of $c_i$ and $c_j$.

As stated in Eq. 5, assuming that $\sum_{u \in V} c_u = B$. Thus, when we fix $c_u$ for all $u \in V \setminus \{i, j\}$, $\sum_{u \in V \setminus \{i, j\}} c_u$ is a constant. Let $B' = B - \sum_{u \in V \setminus \{i, j\}} c_u = c_i + c_j$. In other words, $c_j = B' - c_i$. Combining the other constraints $0 \leq c_i \leq 1$ and $0 \leq c_j \leq 1$ in Eq. 5 we have an equivalent constraint $\text{max}(0, B' - 1) \leq c_i \leq \text{min}(1, B')$.

Thus, in each iteration, increasing $UI(C)$ can be achieved by solving the following optimization problem.

$$
\begin{align*}
\text{maximize} & \quad UI(C) \text{ with respect to } c_i \text{ and } c_j = B' - c_i \\
\text{s.t.} & \quad \text{max}(0, B' - 1) \leq c_i \leq \text{min}(1, B')
\end{align*}
$$

To solve the above optimization problem, we further rewrite $UI(C)$ by fixing $c_u$ for all $u \in$
In Eq. 9, $UI$ we have a new form of the continuous influence maximization problem is given in Algorithm 1.

Thus, Algorithm 1 is a general framework for solving the continuous influence maximization problem.

In Eq. 8 except for $p_i(c_i)$ and $p_j(B' - c_i)$, all terms can be regarded as constants. Therefore, we have a new form of $UI(C)$ with respect to $c_i$ and $c_j = B' - c_i$ as follows.

$$UI(C) = (A_1 + A_2 - A_3 - A_4)p_i(c_i)p_j(B' - c_i) + (A_3 - A_1)p_i(c_i) + (A_4 - A_1)p_j(B' - c_i) + \text{const},$$

where

$$A_1 = \sum_{S \in 2^{V \setminus \{i,j\}}} Pr(S; V \setminus \{i,j\}, C)I(S)$$

$$A_2 = \sum_{S \in 2^{V \setminus \{i,j\}}} Pr(S; V \setminus \{i,j\}, C)I(S \cup \{i,j\})$$

$$A_3 = \sum_{S \in 2^{V \setminus \{i,j\}}} Pr(S; V \setminus \{i,j\}, C)I(S \cup \{i\})$$

$$A_4 = \sum_{S \in 2^{V \setminus \{i,j\}}} Pr(S; V \setminus \{i,j\}, C)I(S \cup \{j\})$$

In Eq. 9, $UI(C)$ only has one variable $c_i$. We take the derivative of $UI(C)$, that is,

$$\frac{dUI(C)}{dc_i} = (A_1 + A_2 - A_3 - A_4)p_i'(c_i)p_j(B' - c_i) - p_i(c_i)p_j'(B' - c_i) + (A_3 - A_1)p_i'(c_i) - (A_4 - A_1)p_j'(B' - c_i)$$

Since $p_i(c_i)$ and $p_j(B' - c_i)$ are both continuously differentiable, the value $c_i \in [\max(0, B' - 1), \min(B', 1)]$ that maximizes the objective function in Eq. 7 must be in one of the following three situations: (1) $c_i = \max(0, B' - 1)$; (2) $c_i = \min(B', 1)$; or (3) $c_i = x$, where $x \in (\max(0, B' - 1), \min(B', 1))$ and $\frac{dUI(C)}{dc_i}|_{c_i=x} = 0$. Thus, we only need to check these three types of points and set $c_i$ to the one that results in the maximum value of $UI(C)$ with respect to $c_i$ and $c_j = B' - c_i$.

Based on the above discussion, the pseudo-code of the coordinate descent algorithm for solving the continuous influence maximization problem is given in Algorithm 1.

We do not assume any specific seed probability function $p_u(c_u)$ and influence function $I(S)$. Thus, Algorithm 1 is a general framework for solving the continuous influence maximization problem.
Algorithm 1 The Coordinate Descent Algorithm

**Input:** Budget $B$, social network $G$, seed probability function $p_u(c_u), \forall u \in V$, and influence function (influence model) $I(S)$

**Output:** Configuration $C$

1: Initialize $C$ such that $C$ satisfies constraints in Eq. 5
2: while not converge do
3:   Pick two variables $c_i$ and $c_j$
4:   $B' \leftarrow c_i + c_j$
5:   Find all $x \in (\max(0, B' - 1), \min(B', 1))$ that $\frac{dUI(C)}{dc_i}\big|_{c_i=x} = 0$
6:   $c_i \leftarrow \arg\max_{c_i \in \{x, \max(0, B'-1), \min(1, B')\}} UIC(C)$
7:   $c_j \leftarrow B' - c_i$
8: return $C$

In Line 3 of Algorithm 1, we do not specify which $c_i$ and $c_j$ should be picked. One heuristic that may help here is to use the partial derivative $\frac{dUI(C)}{dc_i}$ as an indicator. For example, we can pick a variable with a large partial derivative and another variable that has a small partial derivative.

The convergence of Algorithm 1 is guaranteed by the following observations. First, $UI(C) \leq n$, where $n$ is the number of nodes in the social network. Second, after each iteration in Algorithm 1, the updated configuration $C$ ensures that the value of $UI(C)$ is at least as good as the previous one, that is, the value of $UI(C)$ is non-decreasing.

Algorithm 1 approaches a stationary configuration as the limit, which is a necessary condition for finding local optima [2]. Since the objective function $UI(C)$ is not necessarily convex or concave, even when the stationary point is a local optima, it may not be the global optima (note that for non-concave functions, even finding a local maxima is NP-hard). At the same time, because in each iteration the value of our objective cannot be decreased, when taking a configuration $C$ and applying Algorithm 1, we can always find a configuration $C'$ no worse than $C$.

### 5.3 Finding a Good Initial Configuration: Unified Discount Configuration

To run Algorithm 1 effectively, we need a good initial configuration. A practical engineering strategy to design discounts is to offer a unified discount to some users in a social network. That means, for each node $u$ in $G$, $c_u$ is either a predefined value $c$ or 0. Thus, to find a good configuration $C$, we can optimize over the unified discount $c$. Since we have to use up all budget due to the monotonicity of $UI(C)$, to optimize over $c$, we only need to consider situations when $c = \frac{B}{|B|}, \frac{B}{|B|}+1, \ldots, \frac{B}{n}$, which means we only need to try $O(n)$ different values of $c$. Now, the problem becomes how we can find the optimal configuration when $c$ is fixed.

When $c$ is fixed, finding the optimal configuration $C$ is to find the optimal set of users to offer each of them discount $c$. Suppose we choose a set $S$ of users to offer discounts, denote by $Pr(S'; S, c)$ the probability of generating a seed set $S'$ when the unified discount is $c$, that is,

$$Pr(S'; S, c) = \prod_{u \in S'} p_u(c) \prod_{v \in S - S'} (1 - p_v(c)) \quad (12)$$

We define $UI(S; c)$ as the expected influence spread when we offer each user in $S$ a unified
Figure 1: An example illustrating the differences among integer configuration, unified discount configuration and continuous configuration.

discount \(c\). That is,

\[
UI(S; c) = \sum_{S' \in 2^S} Pr(S'; S, c) I(S')
\]  

(13)

We observe the following nice property of \(UI(S; c)\) when \(c\) is fixed.

**Theorem 5** (Monotonicity and submodularity). If \(I(S)\) is monotonic and submodular with respect to \(S\), then \(UI(S; c)\) is also monotonic and submodular with respect to \(S\).

The monotonicity and submodularity of \(UI(S; c)\) with respect to \(S\) imply that, when \(c\) is fixed, we can apply a greedy algorithm to find a set of users \(S\) to offer discounts which can cause influence spread at least \((1 - \frac{1}{e})\) times of the influence spread caused by the optimal set of users \(S^*\). In such a case, when the influence model and seed probability function for each user are given, some efficient influence maximization algorithms \([20, 33, 32]\) can be applied here.

**Example 1.** Fig. 1 is a toy example illustrating the differences between integer configuration, unified discount configuration, and continuous configuration. In Figure 1, the influence model is the IC model and the propagation probabilities along edges are all set to 0.1. Suppose the seed probability functions for the nodes in this graph are all in the form of \(p_u(c_u) = 2c_u - c_u^2\), that is, the users are sensitive to discount. When \(B = 1\), the optimal seeding strategy for DIM is to choose node \(v_1\) as the single seed, which leads to the best integer configuration is \(C_1 = (1, 0, 0, 0, 0)\) and \(UI(C_1) = 1.4\).

If we apply the unified discount strategy, the best unified discount value is 0.2. Correspondingly the best unified discount configuration is \(C_2 = (0.2, 0.2, 0.2, 0.2, 0.2, 0.2)\) and \(UI(C_2) = 1.89216\). If we apply the coordinate descent algorithm and set \(C_2 = (0.2, 0.2, 0.2, 0.2, 0.2)\) as the initial value of configuration, we get a better continuous configuration \(C_3 = (0.38312, 0.15422, 0.15422, 0.15422, 0.15422)\) and \(UI(C_3) = 1.93533\).

### 5.4 The Pool Setting

In real applications, instead of a complete purchase probability curve, sometimes we can only access to the purchase probabilities of users when they are offered certain discounts in a prefixed **discount pool** \(D\). For example, for every user \(u \in V\), we only have values of \(p_u(c_u)\) when \(c_u \in D = \{5\%, 10\%, ..., 100\%\}\). We call this situation the **Pool Setting** of the CIM problem. Correspondingly, we call the situation that we have the complete purchase probability curves of all users the **Curve Setting** of the CIM problem.

Under the pool setting, we do not have the information of derivatives of purchase probability curves. We slightly modify Algorithm 1 by replacing Lines 5 and 6 by an exhaustive search over \(c_u \in \)

---

**Figure 1**

An example illustrating the differences among integer configuration, unified discount configuration and continuous configuration.
\[\max(0, B' - 1), \min(1, B') \cap D.\] Then, the termination condition in Line 2 indicating convergence becomes that we cannot find a pair of users \(i\) and \(j\) such that by adjusting discounts allocated to them \(UI(C)\) can be further improved. To pick two variables \(c_i\) and \(c_j\) to optimize in each iteration, one simple way is to use the Round-Robin method where we try all possible \(O(n^2)\) pairs of \(c_i\) and \(c_j\) in one batch, and keep iterating until the objective \(UI(C)\) cannot be improved in a batch. In practice \(O(n^2)\) may be too large to be feasible. We can only consider nodes whose initial discounts are positive in iterations. In such a case, in every batch we only need \(O(|S|^2)\) pairs to optimize, where \(S\) contains all nodes with positive initial discounts. As we will see in Section 7.3.2, restricting the size of \(S\) is a good idea in practice because by doing so we can avoid the “overfitting” issue. Therefore, \(|S|^2\) is usually not big in practice.

We also slightly modify the unified discount heuristic under the pool setting. Since we can only set the unified discount \(c\) as one element from the discount pool \(D\), it is possible that \(\frac{B}{c}\) is not an integer so by offering the unified discount \(c\) we cannot use up the budget \(B\). In such a case we first greedily allocate \(c\) to \(\lfloor \frac{B}{c} \rfloor\) users, and then allocate the rest budget \(c_l = B - \lfloor \frac{B}{c} \rfloor c < c\) to the user in the rest users (who do not receive the unified discount \(c\)) who can improve the influence spread the most.

### 5.4.1 The Hill Climbing Algorithm

Very recently, following our preliminary conference version [36], Yuan et al. [37] proposed a hill climbing method that can be used to solve the CIM problem under the pool setting. By allowing allocating multiple discounts to each user, Yuan et al. [37] transformed the CIM problem under the pool setting to a submodular set function maximization problem [31]. The Hill Climbing algorithm in [37] can find a \((1 - \frac{1}{e})\)-optimal configuration \(C\) for the CIM problem under the pool setting, where \(c_u \in D\) for every \(u \in V\). For the interest of space, we do not introduce how the Hill Climbing algorithm works in detail.

One may think that by setting the discount pool \(D = \{b, 2b, 3b, ..., 100\%\}\) where \(b\) is the precision parameter, we can solve the CIM problem under the curve setting well as long as \(b\) is small. However, in practice the precision \(b\) cannot be set too small. For the Hill Climbing algorithm in [37], the space complexity is \(O(\frac{n}{b})\) and we need to evaluate the value of \(UI(C)\) for \(O(\frac{Bn}{b})\) times. For large scale networks with big \(n\), if \(b\) is too small, \(O(\frac{n}{b})\) and \(O(\frac{Bn}{b})\) are prohibitive in practice. In [37], the smallest \(b\) was set to 10%. Second, we still need our coordinate descent algorithm and the unified discount heuristic. It is not guaranteed that the solution produced by the Hill Climbing algorithm is always better than the solution generated by our unified discount heuristic. Moreover, we will show by experiment that applying our coordinate descent algorithm on the configuration returned by the Hill Climbing algorithm we can further improve the influence spread (by 2%-5% in experiments), even when the precision parameter \(b\) is set to 1%, a very high precision.

Even under the pool setting, although the Hill Climbing algorithm is \((1 - \frac{1}{e})\)-optimal, we find that our unified discount heuristic plus our coordinate descent algorithm under the pool setting very often produce better solutions. The improvement of our method (unified discount + coordinate descent) compared to Hill Climbing algorithm is 1.5%-4% in experiments.

---

\(^2\)We assume \(c_l\) is always in the discount pool \(D\), which in reality is easy to be satisfied. For example, if \(D = \{1\%, 2\%, ..., 100\%\}\), our assumption holds.
6 CIM and DIM

In this section, we examine the relation between the continuous influence maximization problem (CIM) studied in this paper and the classical and well studied (discrete) influence maximization problem (DIM). Surprisingly, our Coordinate Descent algorithm not only provides us an algorithmic framework for the CIM problem, but also helps us derive relations between CIM and DIM. All results in this section are obtained by utilizing our Coordinate Descent algorithm as an essential building block.

Our first result is that, when the influence function $I(S)$ and the seed probability function $p_u(\cdot)$ satisfy certain conditions, the continuous influence maximization problem and the discrete influence maximization problem share the same optimal solution.

**Theorem 6 (CIM and DIM).** Given an influence function $I(S)$ that is monotonic and submodular, a budget $B$ that is a positive integer, and a seed probability function $p_u(c_u)$ for every node such that $\forall u \in V, \forall c_u \in [0, 1], p_u(c_u) \leq c_u$, the optimal objectives of CIM and DIM are equivalent.

**Corollary 6.1.** Given an influence function $I(S)$ that is monotonic and submodular, an integer budget $B$, if $\forall u \in V, \forall c_u \in [0, 1], p_u(c_u) \leq c_u$, then there exists an integer configuration $C$ that is optimal to CIM.

Theorem 6 also immediately indicates the complexity of the continuous influence maximization problem.

**Corollary 6.2.** If maximizing $I(S)$ is NP-hard, and $I(S)$ is monotonic and submodular, then given a social network $G = (V, E)$, a budget $B$ and the seed probability function $p_u(c_u)$ for each $u \in V$, maximizing $UI(C)$ over $C$ is also NP-hard.

To further understand the significance of Theorem 6 we notice that the seed probability function $p_u(c_u)$ represents how user $u$ is sensitive to discount $c_u$. If $\forall c_u \in [0, 1] p_u(c_u) \leq c_u$, then user $u$ is insensitive to discount. Theorem 6 indicates that, if all users in the network are insensitive to discount, then we would better give free products to some seed users, that is, setting $c_u = 1$, to trigger a sizeable cascade propagation.

Utilizing the proof of Theorem 6 we have an upper bound of $UI(C)$ given a configuration $C$. Define an $l$-seed set a set of $l$ seeds.

**Corollary 6.3 (Upper Bound).** Given an influence function $I(S)$ that is monotonic and submodular, and a discount configuration $C$. Suppose $K = \lceil \sum_{u \in V} p_u(c_u) \rceil$, then $UI(C) \leq I(S^*_K)$, where $S^*_K$ is the optimal $K$-seed set for DIM.

Given a budget $B$, we can solve the following optimization problem to get the maximum expected number of seeds.

\[
\text{maximize} \sum_{u \in V} p_u(c_u) \\
\text{s.t.} \quad 0 \leq c_u \leq 1, \forall u \in V \\
\quad \sum_{u \in V} c_u = B
\]  

(14)

Since a seed probability function $p_u(c_u)$ can be arbitrarily shaped, $\sum_{u \in V} p_u(c_u)$ can be non-concave and non-convex. By assuming that the maximum of Eq. (14) is $K$, where $K$ is decided by seed
probability functions of all nodes and $B$, we have a data dependent approximation ratio of a DIM algorithm and our CIM algorithm (Algorithm 1).

**Theorem 7** (Data-dependent approximation). Given an influence function $I(S)$ that is monotonic and submodular, and an integer budget $B$. Let $K = \max_{C \in \mathbb{C}} \lceil \sum_{u \in V} p_u(c_u) \rceil$, where $\mathbb{C} = \{ C \mid \emptyset \leq C \leq 1, |C|_1 = B \}$. If $S_B$ is a $B$-seed set that is $\alpha$-optimal to the DIM problem with budget $B$, then $I(S_B) \geq \alpha B K U(I(C^*))$, where $U(I(C^*)) = \max_{C \in \mathbb{C}} U(I(C))$. Moreover, if we take $C_B$, the corresponding configuration to $S_B$, as the initial value, then Algorithm 1 is also $\alpha B K$ optimal.

Although under some conditions, CIM and DIM share the same optimal objectives, it can be shown that in some situations it is not the case, particularly when some users are sensitive to discount.

**Example 2.** Consider a social network $G = (V,E)$ where $E = \emptyset$. In other words, $G$ is a graph of $n$ isolated nodes. In the independent influence model or the linear cascade model, if the budget $B = 1$, and for each node $u$, the seed probability function $p_u(c_u) = \sqrt{c_u}$, then the optimal solution for discrete influence maximization is to pick an arbitrary node $u$ and the optimal influence spread is $1$. However, the optimal solution to continuous influence maximization is to assign $\frac{1}{n}$ discount to each node, and the optimal influence spread is $\sqrt{n}$. Thus, not only the optimal solution to discrete influence maximization is not optimal to continuous influence maximization in some cases, but also such solutions to DIM can be arbitrarily bad for CIM as the size of the network becomes very large.

It can be easily shown that CIM can always achieve an influence spread no smaller than DIM. Given a budget $B$, one can first run a DIM algorithm to find a seed set of $\lceil B \rceil$ seeds. Then, by taking the corresponding integer configuration $C$ of the seed set as the initial configuration, after applying the coordinate descent algorithm, a configuration $C'$ that $U(I(C')) \geq U(I(C))$ can be found.

### 7 Implementation of CIM under Triggering Models

All our discussion above assumes that we have an oracle that can return the exact $I(S)$, the influence spread of a given seed set $S$. However, under many influence models, computing $I(S)$ is usually hard and often we can only have an approximation of $I(S)$. This should be considered in implementing our CIM algorithm under a specific influence model.

In this section, we provide specific algorithms for continuous influence maximization under triggering models [16], where the most widely used influence models, such as the Independent Cascade (IC) model [6, 5, 16, 33], the Linear Threshold (LT) model [16, 7, 15] and the Continuous-Time diffusion model [12] are all instances. We give a polling based algorithm and discuss how to avoid the “overfitting” problem similar to the one in machine learning.

#### 7.1 A Recap of the Polling-Based Method

Recently, a polling-based algorithmic framework [1, 33, 32, 26] was proposed for triggering models [16] like the IC model and was shown the most efficient influence maximization algorithm so far. The major idea is to use simulations of propagations of triggering models to find potential influencers of each node, and build a hyper-graph sketch from which we can estimate the influence spread of an arbitrary seed set. Our algorithms are also polling-based algorithms.
Let us first briefly review how a polling method works for influence maximization by taking the Independent Cascade model as an example.

Given a graph $G = \langle V, E \rangle$, a poll is conducted as follows: a node $v \in V$ is picked in random and then we try to find out which nodes are likely to influence $v$. We run a Monte Carlo simulation of an IC propagation $[^{1}] G^T$. Define $pp_{uv}$ the propagation probability of edge $(u, v)$ in $G$. Then in $G^T$, the propagation probability of an edge $(v, u)$ is $pp_{uv}$. Such a “reverse” propagation process from $v$ is used for finding $v$’s potential “influencers”. Suppose the set of nodes reached in this poll is $h$. We call $h$ a random hyper-edge. The intuition of the poll process is that if a node $u$ is highly influential, then the probability that $u$ appears in a random hyper-edge $h$ is high.

For other triggering models, as long as a propagation can be simulated, we can generate a random hyper-edge of the given triggering model. Then the polling method consists of two major steps.

1. Random Hyper-graph Construction. Generate a random hyper-graph $H$ by generating a certain number of random hyper-edges. Note that the nodes in $H$ are still the nodes in $G$. A node $u$ and a hyper-edge $h \in H$ are incident if $u \in h$. Denote by $\deg_H(S)$ the degree of the set of nodes $S$, which is the number of hyper-edges in $H$ incident to at least one node of $S$.

The greater $\deg_H(S)$, the more likely the influence spread of $S$ being high.

2. Maximum Coverage on the Random Hyper-graph. Greedily choose $B$ nodes to add to the seed set $S$ such that the selected node can increase $\deg_H(S)$ the most.

One key property that makes the polling method work well is that when the number of random hyper-edges $m_H$ is fixed, $\frac{\deg_H(S) \times n}{m_H}$ is an unbiased estimation of $I(S)$, the influence spread of $S$ in $G$. Thus, as long as $m_H$ is sufficiently large, the seed set returned by the polling method has good quality guarantees. Tang et al. $[^{33},^{32}]$ illustrated how to estimate the lower bound of $m_H$ that makes the result of the polling method a $(1 - 1/e - \epsilon)$-approximation with probability at least $1 - \frac{1}{n}$ given $\epsilon$.

### 7.2 Polling-based CIM Algorithm

To solve the continuous influence maximization problem, we can employ a similar method that also constructs a random hyper-graph.

**Theorem 8.** Given a graph $G = \langle V, E \rangle$, a triggering model $I$ where a propagation can be simulated and the seed probability functions of all nodes, we generate a random hyper-graph $H$ with $m_H$ hyper-edges generated according to $G$ and $I$. Then for a configuration $C$, $\frac{n \times f_H(C)}{m_H}$ is an unbiased estimation of $UI(C)$, where $f_H(C) = \sum_{h \in H} f_h(C)$ and $f_h(C) = 1 - \prod_{u \in h} (1 - p_u(c_u))$.

According to Theorem 8, we build a random hyper-graph $H$ of $m_H$ hyper-edges. Then we solve the following optimization problem.

$$\begin{align*}
\text{maximize} & \quad f_H(C) = \sum_{h \in H} f_h(C) = \sum_{h \in H} \left[1 - \prod_{u \in h} (1 - p_u(c_u))\right] \\
\text{s.t.} & \quad 0 \leq c_u \leq 1, \forall u \in V \\
& \quad \sum_{u \in V} c_u \leq B
\end{align*}$$

$[^{3}] G^T = \langle V, E^T \rangle$ is the transpose graph of $G = \langle V, E \rangle$ if $\forall (u, v) \in E, (v, u) \in E^T$.
\(f_H(C)\) is an unbiased estimation of \(UI(C)\). In Section 5.2, we proved that the CIM problem (Eq. 3) is NP-hard in general, we now prove that so is the optimization problem in Eq. 15.

**Theorem 9.** Given a hyper-graph \(H\) and the seed probability function \(p_u(c_u)\) for each \(u \in V\), the optimization problem in Eq. 15 is NP-hard in general.

As we pointed out in Section 5.2, \(UI(C)\) is not necessarily concave or convex. It is easy to verify that neither is \(f_H(C)\). Thus, we seek for stationary points as illustrated in Section 5.2, which are necessary conditions of local maxima points (note that finding local maxima of non-concave functions is NP-hard in general [25]). Due to the linear contraints in Eq. 15, a configuration \(C\) is a stationary point if and only if it satisfies the Karush-Kuhn-Tucker (KKT) conditions [2]. By a simple derivation, we have that if \(C\) is a KKT point of Eq. 15, it should satisfy the following.

\[
\nabla_u f_H(C) = \begin{cases} 
\geq \lambda & c_u = 1 \\
= \lambda & 1 > c_u > 0 \\
\leq \lambda & c_u = 0 
\end{cases} \quad \forall u \in V \tag{16}
\]

where \(\nabla_u f_H(C)\) is the partial derivative of \(f_H(C)\) with respect to \(c_u\) and

\[
\nabla_u f_H(C) = p_u(c_u) \sum_{h \in H_u} \prod_{i \in h, i \neq u} (1 - p_i(c_i)) \tag{17}
\]

The condition in Eq. 16 is equivalent to

\[
\max_{u: c_u < 1} \nabla_u f_H(C) \leq \min_{v: c_v > 0} \nabla_v f_H(C) \tag{18}
\]

Our algorithmic framework Algorithm 1 can be used to find KKT points. In every iteration, we pick the discounts of two nodes \(c_u\) and \(c_v\) to optimize, where \(u = \arg \max_{i: c_i < 1} \nabla_i f_H(C)\) and \(v = \arg \min_{j: c_j > 0} \nabla_j f_H(C)\). But this method is time consuming, since in each iteration we need \(O(n)\) time to find \(c_u\) and \(c_v\) to optimize. To avoid this, we adopt a Shrink-and-Expansion strategy which is used in finding dense subgraphs [21].

To illustrate the Shrink-and-Expansion method, we first define a local KKT point on \(S \subseteq V\) as a configuration \(C\) satisfying

\[
ce_u = 0 \quad \text{if} \quad u \notin S
\]

\[
\nabla_u f_H(C) = \begin{cases} 
\geq \lambda & c_u = 1 \\
= \lambda & 1 > c_u > 0 \\
\leq \lambda & c_u = 0 
\end{cases} \quad \forall u \in S \tag{19}
\]

Algorithm 2 describes the Shrink-and-Expansion algorithm. Line 3 is the Shrink stage, because after Line 3 the number of non-zero entries of \(C\) does not get bigger. Line 7 is the expansion stage, where we add nodes whose partial derivatives is greater than the threshold \(t\) with respect to \(S\) for iterations, and the threshold is set to the smallest partial derivative of nodes in \(S\). If in the expansion stage, we do not have any nodes to add to \(S\), clearly the current \(C\) satisfies the KKT condition in Eq. 16 so it is already a KKT point. If \(C\) is only a local KKT point on \(S\) but not a KKT point to Eq. 15 in the expansion stage we must add some nodes to \(S\). Also, after one shrink stage and one expansion stage, the objective \(f_H(C)\) is obviously non-decreasing. Thus, Algorithm 2 always converges and it converges to a KKT point.
Algorithm 2 2-Coordinate Descent Shrink-and-Expansion Algorithm.

Input: A hyper-graph $H$, and an initial configuration $C$

Output: $C$

1: $S \leftarrow \{u \mid c_u > 0\}$
2: while true do
3: Use the 2-Coordinate Descent algorithm and take $C$ as the initial value to find a local KKT point $C^{\text{new}}$ on $S$
4: $C \leftarrow C^{\text{new}}$
5: $S \leftarrow \{u \mid c_u > 0\}$
6: $t \leftarrow \min_{v \in S} \nabla_v f_H(C)$
7: $Z \leftarrow \{i \mid \nabla_i f_H(C) > t\}$
8: if $Z \neq \emptyset$ then
9: $S \leftarrow S \cup Z$
10: else
11: break
12: return $C$

Theorem 10 (KKT Point). Algorithm 2 converges to a KKT configuration $C$.

The advantage of Algorithm 2 is that normally only a small number of nodes are involved in computation, especially when the initial configuration $C$ does not have many non-zero entries. Thus, comparing to directly running line 3 of Algorithm 2 to find a KKT point $C$, which is also a local KKT point on the all vertices set $V$, Algorithm 2 is more efficient.

To find a good initial configuration $C$, we first define

$$f_H(S; c) = \sum_{S' \in 2^S} Pr(S'; S, c) \sum_{h \in H} [1 - \prod_{u \in S' \cap h} (1 - p_u(c))]$$

It is not difficult to verify that when $c$ is fixed, similar to $UI(S; c)$, $f_H(S; c)$ is also monotonic and submodular with respect to $S$. Thus, we can totally apply the Unified Discount heuristic described in section 5.3. However, since we are discussing efficient implementations of CIM algorithms, trying $O(n)$ different values of $c$ in the unified discount heuristic incurs a too high computational cost in practice.

To fix the practical efficiency issue, we only try $\tau$ different values of $c$, where $\tau$ is a predefined constant, when applying the unified discount heuristic to obtain a good initial configuration $C$ for $f_H(C)$. We set a unified discount pool $\{\frac{B}{\lceil \tau B \rceil}, \frac{B}{\lceil \frac{\tau B}{2} \rceil}, \frac{B}{\lceil \frac{\tau B}{4} \rceil}, ..., \frac{B}{\lceil B \rceil}\}$ which contains $\tau$ different unified discounts that spread almost evenly in $[0, 1]$. For a fixed unified discount $c$, we conduct the CELF greedy algorithm [20] to find a set $S_c$ which is $(1 - 1/e)$ optimal for maximizing $f_H(S; c)$ subject to $S \subseteq V$ and $|S| = \frac{B}{c}$. After trying all unified discounts in the unified discount pool, we pick the $c$ which leads to the maximum $f_H(S_c; c)$. Under the pool setting, the $i$-th value $\frac{B}{\lceil \frac{\tau B}{i} \rceil}$ may not be in the discount pool $D$. In such a case, we set $c$ as the discount which is the closest to $\frac{B}{\lceil \frac{\tau B}{i} \rceil}$ in $D$ and apply the unified discount heuristic under the pool setting as described at the end of section 5.4.

Picking a proper $\tau$ is related to the “overfitting” issue introduced in the following of this paper. We will show that to avoid the “overfitting” issue, we can constrain the number of users that we offer non-zero discounts to no more than a predefined number $N$. Given $N$, we set $\tau = \lfloor \frac{N}{B} \rfloor$, since
it is the smallest $\tau$ such that even when we set $c = \frac{B}{|\mathcal{E}_B|}$, which is the smallest value in the unified discount pool, we will not have more than $N$ users to offer discounts.

### 7.3 Deciding Sample Size to Avoid “Overfitting”

Our algorithm needs to build a hyper-graph $H$. How many hyper-edges do we need to achieve a good result? What are the risks if we only sample a small number of random hyper-edges? We answer these questions by making an analogy of machine learning.

#### 7.3.1 An Analogy between Influence Maximization and Machine Learning

We first make an analogy between influence maximization based on the polling method and Empirical Risk Minimization (ERM) machine learning [28].

In machine learning, we have the instance domain $\mathcal{X}$, where an element $x \in \mathcal{X}$ is an instance. Denote by $\Pr(x)$ the probability of obtaining $x$ as a random sample from $\mathcal{X}$. If the domain $\mathcal{X}$ is continuous, then $\Pr(x)$ is the probability density of $x$. The ultimate goal is as follows

$$\max_{x \in \mathcal{X}} \sum_{x \in \mathcal{X}} \Pr(x) g_x(\eta)$$

s.t. $\eta \in \mathcal{H}$

(20)

where $\eta$ is a model and $\mathcal{H}$ is the hypothesis set containing all the models we can use. $g_x(\eta)$ is a function measuring how good is a model $\eta$ on an instance $x$. For example, for a binary classification task, suppose we try to find an optimal linear classifier, and let $x = (x_1, ..., x_n)$ where $x_n \in \{0, 1\}$ is the class label of $x$. Then $g_x(\eta) = 1$ if $(\sum_{i=1}^{n-1} \eta_i x_i + \eta_n) \cdot x_n > 0$ and $g_x(\eta) = 0$ otherwise.

However, solving Eq. (20) is difficult since we normally cannot iterate every instance $x \in \mathcal{X}$. Instead, we can draw a set $\mathcal{T}$ that contains a number of i.i.d. samples, where a sample $x$ is drew according to the probability $\Pr(x)$. We call $\mathcal{T}$ the training set. Adopting the Empirical Risk Minimization (ERM) rule in machine learning, to solve Eq. (20) we solve the following optimization problem.

$$\max_{x \in \mathcal{T}} \sum_{x \in \mathcal{T}} g_x(\eta)$$

s.t. $\eta \in \mathcal{H}$

(21)

We call $\sum_{x \in \mathcal{X}} \Pr(x) g_x(\eta)$ in Eq. (20) the objective and $\sum_{x \in \mathcal{T}} g_x(\eta)$ in Eq. (21) the objective on training data of a machine learning task.

We write the problems of IM and CIM as machine learning problems, as shown in Table 2. For both IM and CIM, the expectation of the objective on training data equals the real objective for any hypothesis. Similar to machine learning, our task in IM or CIM is to choose a good hypothesis with high objective value from the hypothesis set ($\mathcal{S}$ or $\mathcal{C}$), by utilizing the training samples and the objective function on the training data.

One key point of machine learning is, how many training samples do we need to achieve a provable performance guarantee [28]? If we do not have sufficient training samples, we may get overfitting, which means we may choose a hypothesis whose expected objective value on the whole data set is much smaller than its objective value on the training samples.
Deciding the sample size (number of sampled hyper-edges) is also the key point of state-of-the-art influence maximization algorithms based on polling method [1, 33, 32]. Similar to machine learning, if the random hyper-edges (training samples) are not sufficient, we may find a seed set with poor quality and get “overfitting” (the estimated influence of the seed set on the sampled hyper-graph is much greater than its real influence spread).

One major idea of these algorithms is to sample enough hyper-edges to achieve a uniform convergence [28] on $S$ such that

$$\Pr\{\exists S \in S, \left| \frac{\text{deg}_H(S)}{m_H} - \frac{I(S)}{n} \right| > \epsilon \} \leq \delta $$

(22)

When this uniform convergence is achieved, the value $\frac{n \cdot \text{deg}_H(S)}{m_H}$ is a good estimation of $I(S)$ for any $B$-sized seed set $S$. Thus, by maximizing $\frac{n \cdot \text{deg}_H(S)}{m_H}$, we can find a good seed set with near-optimal influence spread (the approximation ratio is decided by $\epsilon$). To achieve Eq. 22, a necessary condition is to sample enough hyper-edges such that for an arbitrary $B$-sized seed set $S$,

$$\Pr\{\left| \frac{\text{deg}_H(S)}{m_H} - \frac{I(S)}{n} \right| > \epsilon \} \leq \left( \binom{n}{B} \right) \delta $$

(23)

Once Eq. 23 is satisfied, by applying the union bound, Eq. 22 holds because the number of different $B$-seed sets (i.e., classifiers) is $\binom{n}{B}$.

### 7.3.2 Sample Size for CIM

We discuss how to decide a proper sample size (number of random hyper-edges) for the CIM problem. One difficulty is that maximizing $f_H(C)$ over a number of sampled hyper-edges is NP-hard as illustrated by Theorem 9. Moreover, $f_H(C)$ is often neither convex nor concave, and so far we can only find a KKT point but we do not know how big the gap between a KKT point and the optimal configuration is. Therefore, unlike influence maximization, where we can get a constant approximation factor $(1 - 1/e - \epsilon)$ ($\epsilon$ is decided by the sample size) with high probability, for the CIM problem, it is hard to figure out the gap between a solution obtained by optimizing $f_H(C)$ over a given number of hyper-edges and the optimal solution. Thus, we seek for a weaker goal that, for the configuration $C$ obtained by optimizing $C$, with high probability, $\frac{f_H(C)}{m_H}$ is close to $\frac{UI(C)}{n}$. Note that, according to the machine learning setting of influence maximization in Section 7.3.1, $\frac{f_H(C)}{m_H}$ is the accuracy of $C$ on the training data and $\frac{UI(C)}{n}$ is the real accuracy of $C$. Formally, we define “overfitting” in CIM as

$$\frac{f_H(C)}{m_H} - \frac{UI(C)}{n} > \epsilon $$

(24)
Note that hyper-edges such that 

\[ C \]

have \[ N \] threshold \[ S \] where the hypothesis set \[ \mathcal{H} \] is to add a constraint to we can play the regularization trick to narrow down the hypothesis set. In CIM, the regularization

Corollary 11.1 (Regularization) \[ \text{Pr} \{ |f_H(C) - UI(C) / n| \leq \epsilon \} \geq 1 - \delta \] (25)

Just achieving Eq. 25 may cause “underfitting” \[ 28\] , where although \[ |f_H(C) - UI(C) / n| \leq \epsilon, \] \[ UI(C) / n \] might not be good enough. This is because fixing the sample size \( m_H \), the smaller \( UI(C) / n \), the higher probability that \[ |f_H(C) - UI(C) / n| \leq \epsilon, \] according to the Chernoff bound \[ 23\]. Thus, just achieving Eq. 25 may lead to find a poor configuration.

We fix the underfitting issue by setting a good configuration \( C \) as the initial configuration of Algorithm \[ 2\]. Taking IM as the baseline, our goal is to find a configuration that is at least as good as the solution produced by the state-of-the-art IM algorithm. We run a state-of-the-art IM algorithm \[ 11, 33, 32, 26\] first to get a discrete configuration \( C_1 \), then we use the sampled hyper-edges, apply the Unified Discount algorithm to find a better configuration \( C_2 \), and apply Algorithm \[ 2\] to find an even better configuration \( C_3 \). Here “better” is with respect to the objective value on sampled hyper-edges. Clearly \( f(C_3) \geq f(C_2) \geq f(C_1) \). Suppose \( f(C_2) = f(C_1) + \delta_2 * m_H \) and \( f(C_3) = f(C_1) + \delta_3 * m_H \). If \( |f_H(C) - UI(C) / n| \leq \epsilon \) holds for \( C = C_1, C_2, C_3 \), then we have

\[
\begin{align*}
\text{Pr}\{f(H(C)) - \frac{UI(C)}{n} \leq \epsilon\} &\leq n\left(\frac{f(H(C))}{m_H} + \epsilon\right) \\
UI(C_2) &\geq UI(C_1) + n(\delta_2 - 2\epsilon) \\
UI(C_3) &\geq UI(C_1) + n(\delta_3 - 2\epsilon)
\end{align*}
\]

Thus, if \( \delta_3 \geq \delta_2 \geq 2\epsilon \), we have \( UI(C_2) \geq UI(C_1) \) and \( UI(C_3) \geq UI(C_1) \). If both \( \delta_2 \) and \( \delta_3 \) are no greater than \( 2\epsilon \), to avoid the risk of getting a configuration \( C \) such that \( UI(C) \leq UI(C_1) \), we can just conservatively pick \( C_1 \) as our final configuration. Therefore, taking IM as the baseline, we can avoid the underfitting issue where our final configuration is at least as good as the solution of IM.

One necessary condition of Eq. 25 is the uniform convergence on \( C \), that is, we need to ensure

\[ \text{Pr}\{\forall C \in \mathcal{C}, |f_H(C) - UI(C) / n| \leq \epsilon\} \geq 1 - \delta \] (27)

where \( \mathcal{C} = \{ C \mid 0 \leq C \leq 1, |C|_1 = B \} \) is our hypothesis set. Unlike influence maximization, where the hypothesis set \( \mathcal{S} \) contains only \( \binom{B}{n} \) hypotheses, \( \mathcal{C} \) is uncountably infinite. This seems to be a trouble for us to apply the union bound, but we show that we can still achieve the uniform convergence in Eq. 25 by sampling a finite number of hyper-edges.

Theorem 11 (Uniform Convergence). By sampling \( m_H \geq \frac{3(n \ln 2 + \ln \frac{2}{\epsilon})}{\epsilon^2} \) hyper-edges, we have \[ \text{Pr}\{\forall C \in \mathcal{C}, |f_H(C) - UI(C) / n| \leq \epsilon\} \geq 1 - \delta, \] where \( \mathcal{C} = \{ C \mid 0 \leq C \leq 1, |C|_1 = B, |C|_0 \leq N \} \).

Similar to many machine learning methods, if we do not want to sample too many hyper-edges, we can play the regularization trick to narrow down the hypothesis set. In CIM, the regularization is to add a constraint to \( C \) such that the number of non-zero entries \( |C|_0 \) is no greater than a threshold \( N \). Therefore, the hypothesis set becomes \( \mathcal{C}_N = \{ C \mid 0 \leq C \leq 1, |C|_1 = B, |C|_0 \leq N \} \). Note that \( |\mathcal{C}_N| = \sum_{i=B}^{N} \binom{n}{B} \). Similar to Theorem 11, we have the following corollary.

Corollary 11.1 (Regularization). By setting the number of hyper-edges \( m_H \geq \frac{3(N + N \ln 2 + \ln \frac{2}{\epsilon^2})}{\epsilon^2} \), we have \[ \text{Pr}\{\forall C \in \mathcal{C}_N, |f_H(C) - UI(C) / n| \leq \epsilon\} \geq 1 - \delta, \] where \( \mathcal{C}_N = \{ C \mid 0 \leq C \leq 1, |C|_1 = B, |C|_0 \leq N \} \).
Algorithm 3 Optimization-and-Estimation Strategy

Input: Two hyper-graphs \( H_1 \) and \( H_2 \) that both have \( m_H \) hyper-edges

Output: \( C \)

1: Set \( M \) to \( 3(\ln 2 + \ln \frac{1}{\epsilon}) \) (without regularization) or \( 3(N + N \ln \frac{2}{\epsilon} + \ln \frac{2}{\epsilon}) \) (with regularization)
2: while \( m_H \leq M \) do
3: \( L \leftarrow |f_{H_2}(C)| \)
4: Optimize \( f_{H_1}(C) \) on \( H_1 \) to get \( C \)
5: Find the smallest index \( i_L \) such that \( \sum_{i=1}^{i_L} f_{h_i}(C) \geq L \), where \( h_1, h_2, ..., h_{i_L} \in H_2 \)
6: \( e \leftarrow \max(\frac{|f_{H_1}(C)|}{m_H} - L, (1 + \epsilon_1)\frac{|f_{H_1}(C)|}{m_H} - L) \)
7: if \( e \leq \epsilon \) then
8: \( \text{break} \)
9: else
10: \( H_1 \leftarrow H_1 \cup H_2 \)
11: Sample \( |H_1| = 2m_H \) random hyper-edges to rebuild \( H_2 \)
12: \( m_H \leftarrow 2 \times m_H \)
13: return \( C \)

When we have the regularization with parameter \( N \), we slightly modify Algorithm 3. In the expansion stage (Line 7), if \( |Z| > N - |S| \), we truncate \( Z \) by only keeping the top \( N - |S| \) nodes with respect to their partial derivatives in \( Z \), and in the next while loop, we only do the shrink stage and return the \( C \) obtained.

Theorem 11 and Corollary 11.1 provide necessary conditions for avoiding overfitting in CIM. These two bounds of \( m_H \) may be loose. Thus, in practice, we use a similar strategy as the state-of-the-art IM algorithm in [20], where we keep two hyper graphs \( H_1 \) and \( H_2 \) that both have \( m_H \) hyper-edges. We maximize \( f_{H_1}(C) \) to get \( C \) on \( H_1 \), and estimate \( \frac{U(C)}{n} \) using \( H_2 \). Suppose \( |f_{H_2}(C)| = L \). Note that \( f_{H_2}(C) = \sum_{h \in H_2} f_h(C) \), where \( f_h(C) = 1 - \prod_{u \in h} (1 - p_u(c_u)) \). We find the smallest index \( i_L \) of hyper-edges in \( H_2 \) such that \( \sum_{i=1}^{i_L} f_{h_i}(C) \geq L \), where \( h_i \) is the \( i \)-th hyper-edge of \( H_2 \). Since \( f_{h_i}(C) \) is a random variable (due to the randomness of \( h \)) distributed in \([0, 1]\) and \( E[f_h(C)] = \frac{U(C)}{n} \), we can apply the Stop-Rule Theorem [9] and obtain that \( \frac{C}{i_L} \) is an \((\epsilon_1, \delta)\) estimation of \( \frac{U(C)}{n} \), where \( 1 + (1 + \epsilon_1)\frac{4(1 - \epsilon)\ln \frac{2}{\epsilon}}{\epsilon^2} \). Thus, with probability at least \( 1 - \delta \), we have \( \frac{U(C)}{n} \in \left[ \frac{C}{i_L}, (1 + \epsilon_1)\frac{C}{i_L} \right] \). Then we check if the maximum possible estimation error of \( \frac{U(C)}{m_H} \), \( \max(\frac{|f_{H_1}(C)|}{m_H} - L, (1 + \epsilon_1)\frac{|f_{H_1}(C)|}{m_H} - L) \), is less than \( \epsilon \). If it is, we are done. If not, we set \( H_1 = H_1 \cup H_2 \) and we rebuild \( H_2 \) by sampling \( |H_1| = 2m_H \) random hyper-edges. Then we redo the above process until the maximum possible estimation error of \( \frac{U(C)}{m_H} \) is less than \( \epsilon \), or \( |H_1| \) exceeds the value of \( m_H \) in Theorem 11 or Corollary 11.1. We describe the Optimization-and-Estimation Strategy in Algorithm 3.

The input of Algorithm 3 can be obtained by the state-of-the-art algorithm SSA [20] for influence maximization, where SSA exactly outputs two hyper-graphs \( H_1 \) and \( H_2 \) that have the same size. The seed set returned by SSA can be transformed to an integer configuration and be used to find a good initial configuration. Moreover, the result of SSA can help us control a relative error (note that our goal Eq. 23 controls an absolute error). Suppose the integer configuration returned by SSA is \( C_1 \), we just need to set \( \epsilon = O(\frac{f_{H_1}(C_1)}{m_H}) \) because it is guaranteed that \( f_{H_1}(C) \geq f_{H_1}(C_1) \).
where $C$ is obtained in Line 3 in Algorithm 3. The coefficient hidden in $O\left(\frac{f_{H_1}(C_1)}{m_H}\right)$ is decided by the relative error of SSA, our desired relative error and the parameter $\delta$ in Eq. 25.

8 Empirical Evaluation

To examine the effectiveness and efficiency of our methods, in this section, we report experiments on four real networks with synthesized seed probability functions to test our proposed methods. The experiment results show that the continuous influence maximization strategy can significantly improve influence spread without incurring dramatic extra overheads compared to discrete influence maximization.

8.1 Experimental Settings

We ran our experiments on four real network data sets that are publicly available in SNAP (http://snap.stanford.edu/data/index.html). Table 3 shows the details of the four data sets. All networks are treated as directed graphs, which means if a network is undirected, every undirected edge $(u,v)$ is processed as two directed edges $(u,v)$ and $(v,u)$.

In our experiments, we adopted the Independent Cascade (IC) model as the influence model, which is the most widely used triggering model in literature [6, 5, 16, 33, 26]. Following the most popular settings of the IC model [6, 5, 16, 33, 26], we set the propagation probability of a directed edge $(u,v)$ to $\frac{1}{\text{degree}(v)}$.

For seed probability functions, unfortunately we do not have access to any such real data sets for the purpose of experiments. Thus, we used synthesized seed probability functions. Given a

<table>
<thead>
<tr>
<th>Network</th>
<th>$n$</th>
<th>$m$</th>
<th>Average Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>wiki-Vote</td>
<td>7,115</td>
<td>103,689</td>
<td>14.6</td>
</tr>
<tr>
<td>ca-AstroPh</td>
<td>18,772</td>
<td>396,220</td>
<td>21.1</td>
</tr>
<tr>
<td>com-dblp</td>
<td>317,080</td>
<td>2,099,732</td>
<td>6.6</td>
</tr>
<tr>
<td>com-LiveJornal</td>
<td>3,997,962</td>
<td>69,362,378</td>
<td>17.3</td>
</tr>
</tbody>
</table>

Table 3: Datasets
network $G = \langle V, E \rangle$, we randomly assigned $p_u(c_u) = 2c_u - c_u^2$ to a $p_1$ portion of nodes, which means that those users were sensitive to discount. Then a $p_2$ portion of nodes were assigned with $p_u(c_u) = c_u$, and a $p_3$ portion of nodes were assigned with $p_u(c_u) = c_u^2$. These users (nodes) were insensitive to discount. We maintain $p_1 + p_2 + p_3 = 1$. Figure 2 shows the curves of the two seed probability functions as well as the function $p_u(c_u) = c_u$ as the reference. In our experiments, we set $P = (p_1, p_2, p_3)$ as $(0.85, 0.1, 0.05), (0.75, 0.15, 0.1)$ and $(0.65, 0.2, 0.15)$. For each $P$, we ran 20 independent experiments by randomly assigning seed probability functions to nodes. All results reported are the averages taken over the results of 20 independent experiments.

We also tested the pool setting described in Section 5.4 although the aim of this paper is to settle the CIM problem under the curve setting. We still used the 3 curves $p_u(c_u) = 2c_u - c_u^2$, $p_u(c_u) = c_u$ and $p_u(c_u) = c_u^2$, and portion of each curve is set the same as described above. We set the discount pool $D = \{1\%, 2\%, 3\%, ..., 100\%\}$.

We compare a series of algorithms: discrete influence maximization (IM), the unified discount heuristic for finding a good initial configuration (UC), applying the coordinate descent algorithm based on the unified discount heuristic (UC+CD), the hill climbing algorithm (HC), applying the coordinate descent algorithm based on the result of the hill climbing algorithm (HC+CD), and the one that picks the better one between UC+CD and HC+CD (CD). Note that except for IM, all other algorithms are CIM algorithms since they can allocate non-integer discounts to users. We adopted the state-of-the-art SSA [26] algorithm as the IM implementation. According to Theorem 7, for the result of IM, the better approximation it is to DIM, the better approximation it is to CIM. Thus, we implemented the IM algorithm by setting $\epsilon = 0.03$ and $\delta = \frac{1}{n}$, which means with probability at least $1 - \frac{1}{n}$, the result of IM is at least 60%-optimal to DIM. For the other algorithms compared, we used our Algorithm 3 in section 7.3.2 to avoid the “overfitting” issue. The two piles of RR sets generated by the SSA [26] algorithm are used as the input of Algorithm 3. We played the regularization trick and set $N = 20B$, which means we offer non-zero discounts to at most $20B$ users. The absolute error $\epsilon$ of $\frac{|I_{HC}(C) - U(H(C)|}{\mu H}$ in Eq. 24 was set to 0.01. For our unified discount heuristic (UC), as introduced in at the end of section 7.2, we set $\tau = 20$ since $N = 20B$.

For the hill climbing algorithm, as illustrated in section 5.4, there is an important parameter $b$. We tried two values of $b$, 0.05 and 0.01. Thus, the four algorithm related to the hill climbing algorithms are denoted by HC(0.05), HC(0.01), HC(0.05)+CD and HC(0.01)+CD. Note that as discussed in section 5.4, $b$ is usually set not to small and in [37] the smallest $b$ is 0.1. Thus, 0.05 or 0.01 are practically very high precision for the parameter $b$ in the hill climbing algorithm.

All algorithms were implemented in C# and ran on an Windows 10 computer with Intel(R) Core(TM) i7-3770 3.40GHz CPU, 32GB main memory.

8.2 Effectiveness

Fig. 3 and Fig. 4 show the influence spread of each algorithm under different settings of parameters. Note that in our Algorithm 3 described in Section 7.3.2 we have two hyper-graphs. The second one is for testing the influence spread of a given configuration. By the Stop Rule Theorem [9], we find that the second hyper-graph estimates influence spreads of configurations obtained by different algorithms accurately. With very high probability $(1 - O(\frac{1}{n}))$, the relative error of estimations

\footnote{For the largest dataset Live-Journal, our main memory (32G) cannot afford setting $b = 0.01$ for the hill climbing algorithm. Thus, we set $b = 0.02$ instead, and the corresponding algorithms are denoted by HC(0.02) and HC(0.02)+CD.}
never exceeds 2%. From the results we find that all CIM algorithms can significantly increase the expected influence spread compared to discrete influence maximization (IM).

Under the curve setting, almost every time the three methods UC+CD, HC(0.05)+CD and HC(0.01)+CD achieve very close influence spreads and are the best. Moreover, setting \( b = 0.01 \), a higher precision, does not help improve the effectiveness of the hill climbing algorithm. The influence spread of HC(0.01) often is pretty much the same as the influence spread of HC(0.05), and sometimes even slightly worse. We have similar observation on the effectiveness of HC(0.01)+CD and HC(0.05)+CD. Another important observation is that the coordinate descent algorithm can always refine the input configuration to achieve an obviously better influence spread, no matter the input configuration is obtained by UC, HC(0.05) or HC(0.01). The improvement according to Fig. 3 is 2% – 5%.

Under the pool setting, in almost every case the UC+CD algorithm achieves the best influence spread. Although HC(0.05) and HC(0.01) can obtain better configurations than the UC algorithm, after the refinement of the CD algorithm (under the pool setting), the configuration returned by UC+CD often becomes the best. The improvement according to Fig. 4 is normally 1% – 4%. Moreover, under the pool setting, the CD algorithm cannot improve the configuration obtained by the hill climbing algorithm (HC(0.05) and HC(0.01)) much. The results demonstrate that under the pool setting, although the hill climbing has an approximation ratio, our UC+CD algorithm is a highly competitive solution in practice.

8.3 Efficiency and Scalability

We also tested the efficiency and the scalability of algorithms compared and report the results in Fig. 5 and Fig. 6. The IM algorithm is always the fastest, since all other algorithms need to run the IM algorithm (the SSA [26]) first and take the two piles of RR sets generated by IM as the input to run the Optimization-and-Estimation Strategy (Algorithm 3). However, the running time of different algorithms does not differ much. The most time-consuming algorithm (sometimes UC+CD and sometimes HC(0.01)+CD) only takes around 1.5 to 3 times of the time spent by the most efficient algorithm IM. This is because the two random hyper graphs produced by the IM algorithm are always enough for avoiding the “overfitting” issue in the other CIM algorithms. Thus, every time we broke the while loop in Algorithm 3 in the first round.

9 Conclusions

In this paper, we propose to offer users in social networks discounts rather than free products to trigger social cascades. We model the continuous influence maximization problem. Some key properties of the continuous influence maximization problem are studied and a coordinate descent framework is devised. Based on this framework, we prove that under certain conditions the continuous influence maximization problem and the original influence maximization problem share the same optimal solutions. We also demonstrate that there is a data dependent approximation ratio for our solution, where the ratio is decided by the approximation ratio of a traditional influence maximization algorithm (used for finding a good initial configuration) and all seed probability functions. We then develop methods for implementing the CIM algorithm under triggering models. An analogy between polling-based algorithms and machine learning is discussed. Inspired by this analogy, we point out that there are problems similar to the overfitting issue in machine learning and
Figure 3: Influence spread under curve setting
Figure 4: Influence spread under pool setting
(a) wiki-Vote $P = (0.85, 0.1, 0.05)$

(b) wiki-Vote $P = (0.75, 0.15, 0.1)$

(c) wiki-Vote $P = (0.65, 0.2, 0.15)$

(d) ca-AstroPh $P = (0.85, 0.1, 0.05)$

(e) ca-AstroPh $P = (0.75, 0.15, 0.1)$

(f) ca-AstroPh $P = (0.65, 0.2, 0.15)$

(g) com-dblp $P = (0.85, 0.1, 0.05)$

(h) com-dblp $P = (0.75, 0.15, 0.1)$

(i) com-dblp $P = (0.65, 0.2, 0.15)$

(j) com-LiveJournal $P = (0.85, 0.1, 0.05)$

(k) com-LiveJournal $P = (0.75, 0.15, 0.1)$

(l) on com-LiveJournal $P = (0.65, 0.2, 0.15)$

Figure 5: Running time under curve setting
Figure 6: Running time under pool setting

(a) wiki-Vote $P = (0.85, 0.1, 0.05)$

(b) wiki-Vote $P = (0.75, 0.15, 0.1)$

(c) wiki-Vote $P = (0.65, 0.2, 0.15)$

(d) ca-AstroPh $P = (0.85, 0.1, 0.05)$

(e) ca-AstroPh $P = (0.75, 0.15, 0.1)$

(f) ca-AstroPh $P = (0.65, 0.2, 0.15)$

(g) com-dblp $P = (0.85, 0.1, 0.05)$

(h) com-dblp $P = (0.75, 0.15, 0.1)$

(i) com-dblp $P = (0.65, 0.2, 0.15)$

(j) com-LiveJournal $P = (0.85, 0.1, 0.05)$

(k) com-LiveJournal $P = (0.75, 0.15, 0.1)$

(l) on com-LiveJournal $P = (0.65, 0.2, 0.15)$
devise methods to avoid the “overfitting” issue in CIM. The experiment results demonstrate that our methods can improve influence spreads significantly compared to traditional influence maximization and are more robust to perturbations of seed probability functions than a hill climbing algorithm, while the extra running time over the baselines is not much.

We believe that this work opens a new direction for future work. For example, in the Unified Discount (UC) heuristic for finding a good initial configuration, we conduct a brute force search to find the optimal discount $c$. Is there a better algorithm to search $c$? Another interesting direction is minimizing the budget of our continuous seeding strategy to cover a given portion of users in a social network. While minimizing budget under integer seeding strategy can be easily obtained by slightly modifying the greedy algorithm for influence maximization, it is far from trivial to design a new algorithm for our continuous seeding strategy.

References


Appendix: Proofs of Theoretical Results

Proof of Theorem 1

Proof. We prove by a simple reduction from computing $I(S)$. For any $S$, we can make a configuration $C$ such that $c_u = 1$ if $u \in S$ and $c_u = 0$ otherwise. Clearly we have $UI(C) = I(S)$. Thus, if computing $I(S)$ is $\#P$-hard, so is computing $UI(C)$. \hfill \square

Proof of Theorem 2

Proof. Recall that $UI(C) = \sum_{S \in 2^V} Pr(S; V, C)I(S)$. $\forall S \in 2^V$, we have $0 \leq I(S) \leq n$. We can estimate $UI(C)$ by a Monte Carlo method. By applying the Hoeffding bound, we have

$$Pr(|\hat{U}I(C) - U(C)| \geq \epsilon U(C)) \leq 2e^{-\frac{2\epsilon^2 \sum_{s \in V} R(s)}{2n^2}},$$

where $R$ is the number of MC simulations. Since $UI(C) \geq \sum_{u \in V} p_u(c_u)$, to achieve the goal that $Pr(|\hat{U}I(C) - U(C)| \geq \epsilon U(C)) \leq \delta$, we can set $R \geq \frac{2\epsilon^2 \sum_{u \in V} (n(\epsilon) + 1)}{2n^2 \epsilon^2}$. \hfill \square

Proof of Theorem 3

Proof. If there is a FPRAS for estimating $I(S)$, in $O(\text{POLY}(\frac{n}{\epsilon} \ln \frac{1}{\delta}))$ time we can obtain an $(\epsilon, \delta')$ approximation of $I(S)$. If we set $\delta' = \frac{\delta}{O(\text{POLY}(\frac{n}{\epsilon} \ln \frac{1}{\delta}))}$, the time we need is $O(\text{POLY}(\frac{n}{\epsilon} \ln \frac{1}{\delta})) = O(\text{POLY}(\frac{n}{\epsilon} \ln \frac{1}{\delta})).$

According to Theorem 2 if we can access the value of $I(S)$, we can have an $(\epsilon', \delta_1)$ approximation of $UI(C)$ by calling the oracle $O(\text{POLY}(\frac{n}{\epsilon} \ln \frac{1}{\delta}))$ times. Suppose $R = O(\text{POLY}(\frac{n}{\epsilon} \ln \frac{1}{\delta}))$, and we have $R$ randomly generated seed sets $\{S_1, S_2, ..., S_R\}$. Let $\hat{U}I(C) = \sum_{i=1}^{R} \hat{I}(S_i)$ be the estimation of $I(S_i)$ obtained by the FPRAS in $O(\text{POLY}(\frac{n}{\epsilon} \ln \frac{1}{\delta}))$ time, and $\hat{U}I(C) = \sum_{i=1}^{R} \hat{I}(S_i)$. Clearly, $Pr(|\hat{U}I(C) - U(C)| > \epsilon U(C)) < \delta_1$, and $Pr(|\hat{U}I(C) - \hat{U}I(C)| > \epsilon \hat{U}I(C)) < \delta_1$ (by Union Bound). Applying the union bound, we have that $Pr(|\hat{U}I(C) - U(C)| < \epsilon (2 + \epsilon) U(C)) > 1 - 2\delta_1$.

Setting $\epsilon (2 + \epsilon) = \epsilon$ and $2\delta_1 = \delta$, which means $\epsilon' = \sqrt{1 + \epsilon} - 1$ and $\delta_1 = \frac{\delta}{2}$, we can obtain an $(\epsilon, \delta)$ approximation of $UI(C)$ in $O(\text{POLY}(\frac{n}{\epsilon} \ln \frac{1}{\delta})) \times O(\text{POLY}(\frac{n}{\epsilon} \ln \frac{1}{\delta})) = O(\text{POLY}(\frac{n}{\epsilon} \ln \frac{1}{\delta}))$ time. Note that $\frac{1}{\epsilon'} = \frac{1}{\sqrt{1 + \epsilon} - 1} = \frac{\sqrt{1 + \epsilon} + 1}{\epsilon} \leq \frac{3}{\epsilon} = O(\frac{1}{\epsilon})$, and $\ln \frac{1}{\delta} = O(\ln \frac{1}{\epsilon})$. So in $O(\text{POLY}(\frac{n}{\epsilon} \ln \frac{1}{\delta}))$ time we can have an $(\epsilon, \delta)$ approximation of $UI(C)$.

Proof of Lemma 1

Proof. Because $p_u(c_u)$ is monotonic with respect to $c_u$, we have $p_u(c_u^1) \geq p_u(c_u^2)$. Thus, $p_u(c_u^1) - p_u(c_u^2) = \alpha \geq 0$. We have

$$UI(C) = \sum_{S \in 2^V \backslash \{u\}} Pr(S; V \backslash \{u\}, C)I(S)[1 - p_u(c_u)] + \sum_{S \in 2^V \backslash \{u\}} Pr(S; V \backslash \{u\}, C)I(S \cup \{u\})p_u(c_u)$$
Therefore,

\[
UI(C_1) - UI(C_2) = \sum_{S \in 2^V \setminus \{u\}} Pr(S; V \setminus \{u\}, C_1)I(S \cup \{u\})\alpha - \sum_{S \in 2^V \setminus \{u\}} Pr(S; V \setminus \{u\}, C_2)I(S)\alpha \\
= \sum_{S \in 2^V \setminus \{u\}} \alpha Pr(S; V \setminus \{u\}, C_2)[I(S \cup \{u\}) - I(S)]
\]

Due to the monotonicity of \(I(S), I(S \cup \{u\}) - I(S) \geq 0\) and \(UI(C_1) - UI(C_2) \geq 0\). Thus, we have \(UI(C_1) \geq UI(C_2)\).

\[\square\]

**Proof of Theorem 6**

**Proof.** The major idea of our proof is to show that, for an arbitrary feasible configuration \(C\), there is an integer configuration \(C'\) such that \(UI(C') \geq UI(C)\). As \(p_u(c_u) \leq c_u\) stated in the theorem, we consider two cases.

First, we consider the situation where \(\forall u \in V, p_u(c_u) = c_u\). For any feasible configuration \(C\), in Line 3 of Algorithm 1 if \(C\) contains a component \(c_i\) that is not an integer, since \(B\) is an integer, \(C\) must contains another component \(c_j\) (\(i \neq j\)) such that \(c_j\) is not an integer, either. Therefore, we can always pick non-integers \(c_i\) and \(c_j\). Then, we optimize over \(c_i\) and \(c_j\) by solving the optimization problem in Eq. 7. Due to Eq. 7 we have

\[
UI(C) = (A_1 + A_2 - A_3 - A_4)c_i(B' - c_i) + (A_3 - A_1)c_i + (A_4 - A_1)(B' - c_i)
\]

which is a quadratic form of \(c_i\) and the coefficient of the quadratic term is \(-(A_1 + A_2 - A_3 - A_4)\). Since \(I(S)\) is submodular, we have \(I(S \cup \{i, j\}) - I(S \cup \{i\}) \leq I(S \cup \{j\}) - I(S)\). Thus, \((A_1 + A_2 - A_3 - A_4) \leq 0\), which means \(UI(C)\) is a convex function with respect to \(c_i\). Therefore, the value of \(x\) that makes \(\frac{dUI(C)}{dc_i}\big|_{c_i=x} = 0\) is the global minimum. Since we are interested in only the maximum, we can ignore the root of \(\frac{dUI(C)}{dc_i} = 0\) completely. This means that, after optimization, \(c_i\) must be either \(\text{max}(0, B' - 1)\) or \(\text{min}(B', 1)\).

Let us examine the value of \(c_i\) and \(c_j\) after optimization under different situations of \(B'\). There are two possible cases.

- If \(B' \geq 1\), then \(B' - 1 \leq c_i \leq 1\). After optimization over \(c_i\) and \(c_j\), if \(c_i = B' - 1\), then \(c_j = 1\). If \(c_i = 1\), then \(c_j = B' - 1\).
- If \(B' < 1\), then \(0 \leq c_i \leq B'\). After optimization over \(c_i\) and \(c_j\), if \(c_i = 0\), then \(c_j = B'\). If \(c_i = B'\), then \(c_j = 0\).

Thus, after optimization over \(c_i\) and \(c_j\), at least one variable of \(c_i\) and \(c_j\) takes an integer value. In other words, after one iteration we eliminate at least one non-integer \(c_u\). Apparently, after at most \(n\) iterations we can make all \(c_u\)'s integers. Since in every iteration the objective function does not decrease, the final integer configuration \(C'\) can achieve an influence spread no smaller than the initial configuration. Therefore, we only need to consider integer configurations, which means CIM degenerates into DIM.

Second, we consider the situation when there exists at least one node \(u\) such that \(p_u(c_u) < c_u\). In other words, we consider for each \(u\), \(p_u(c_u) \leq c_u\). Let \(\tilde{p}_u(c_u)\) be the seed probability function such that for each \(u\), \(\tilde{p}_u(c_u) = c_u\). Denote by \(\tilde{UI}(C)\) be the influence spread using \(\tilde{p}_u(c_u)\) with respect to
configuration \( C \). For each \( u \), due to the assumption that \( p_u(\cdot) \) is continuous, we have \( \bar{c}_u \) such that 
\[
p_u(c_u) = \bar{p}_u(c_u) \leq c_u = \bar{p}_u(c_u).
\]
Consider two configurations \( C = (c_1, \ldots, c_n) \) and \( \bar{C} = (\bar{c}_1, \ldots, \bar{c}_n) \).
Clearly, \( \bar{C} \preceq C \). Due to the monotonicity of \( UI(C) \), we have 
\[
UI(C) = \bar{UI}(\bar{C}) \leq \bar{UI}(C).
\]
As the first case, we already prove that \( p \) is the seed probability function of \( u \) for every \( u \in V \). We use notations in Section 3 \( UI(C) \) and \( I(S) \), to denote the expected influence of a configuration \( C \) and the influence of a seed set \( S \) under the setting that \( p_u(c_u) \) is the seed probability function of \( u \) for every \( u \in V \). Let \( C' = (p_1(c_1), p_2(c_2), \ldots, p_n(c_n)) \) and with out loss of generality, assume \( \sum_{u \in V} p_u(c_u) \) equals an integer \( K \). Now, for every \( u \in V \), we replace the seed probability function of \( u \) by a new one, \( p'_u(c_u) = c_u \). Denote by \( \bar{UI}(C') \) the expected influence of \( C' \), and \( \bar{I}(S) \) the influence spread of a seed set \( S \), under the new seed probability function setting. Clearly, \( \bar{UI}(C') = UI(C) \). Also, \( \bar{I}(S) = I(S) \) for every \( S \subseteq V \). According to the proof of Theorem 6 \( \bar{UI}(C') \) is not optimal under the budget constraint \( B = \sum_{u \in V} p_1(c_1) = K \) and \( \bar{UI}(C') \leq I'(S'_K) \), where \( S'_K^* \) is the optimal \( K \)-seed set for the DIM problem. Thus, \( UI(C) = \bar{UI}(C') \leq I'(S'_K) = I(S'_K) \).

Proof of Theorem 5

Proof. The monotonicity can be immediately proved using Theorem 4. Next, we show the submodularity of \( UI(S; c) \).

Suppose we have two sets \( S_1 \) and \( S_2 \) such that \( S_1 \cup \{v\} = S_2 \). Let \( u \) be a node such that \( u \notin S_2 \). Then,
\[
UI(S_1 \cup \{u\}; c) - UI(S_1; c) = \sum_{S \subseteq S_1} Pr(S; S_1, c) \left( p_u(c)I(S \cup \{u\}) + (1 - p_u(c))I(S) \right) - \sum_{S \subseteq S_1} Pr(S; S_1, c)I(S) \\
= p_u(c) \sum_{S \subseteq S_1} Pr(S; S_1, c) \left( I(S \cup \{u\}) - I(S) \right)
\]

We also have
\[
UI(S_2 \cup \{u\}; c) - UI(S_2; c) = \sum_{S \subseteq 2^{S_1}} Pr(S; S_1, c) \left( p_v(c)p_u(c)I(S \cup \{u, v\}) + p_v(c)(1 - p_u(c))I(S \cup \{v\}) + p_u(c)(1 - p_v(c))I(S \cup \{u\}) + (1 - p_u(c))(1 - p_v(c))I(S) \right) - \\
\sum_{S \subseteq 2^{S_1}} Pr(S; S_1, c) \left( p_v(c)I(S \cup \{v\}) + (1 - p_v(c))I(S) \right) \\
= \sum_{S \subseteq 2^{S_1}} Pr(S; S_1, c) \left( p_u(c)p_v(c) \left( I(S \cup \{u, v\}) - I(S \cup \{v\}) \right) + p_u(c)(1 - p_v(c)) \left( I(S \cup \{u\}) - I(S) \right) \right)
\]
Due to the submodularity of $I(S)$, $I(S \cup \{u, v\}) - I(S \cup \{v\}) \leq I(S \cup \{u\}) - I(S)$. Therefore,

$$UI(S_1 \cup \{u\}; c) - UI(S_1; c)$$

$$\leq \sum_{S \in 2^{S_1}} Pr(S; S_1, c)\left(p_u(c)p_v(c)(I(S \cup \{u\}) - I(S)) + p_u(c)(1 - p_v(c))(I(S \cup \{u\}) - I(S))\right)$$

$$= p_u(c) \sum_{S \in 2^{S_1}} Pr(S; S_1, c)(I(S \cup \{u\}) - I(S))$$

$$= UI(S_1 \cup \{u\}; c) - UI(S_1; c)$$

That is, we prove

$$UI(S_1 \cup \{u\}; c) - UI(S_1; c) \geq UI(S_2 \cup \{u\}; c) - UI(S_2; c).$$

By a simple induction, we can show that, if $S \subseteq T$ and $u \notin T$, $UI(S \cup \{u\}; c) - UI(S; c) \geq UI(T \cup \{u\}; c) - UI(T; c)$, which means $UI(S; c)$ is submodular with respect to $S$.

**Proof of Theorem 7**

Proof. Denote by $S_B^*$ and $S_K^*$ the optimal $B$-seed set and the optimal $K$-seed set for DIM. By Corollary 6.3 and the monotonicity of $I(S)$, we can immediately get that $UI(C^*) \leq I(S_K^*)$. We can run the greedy algorithm on $S_K^*$ to sort nodes in it. Suppose $S_K^* = \{u_1, u_2, ..., u_K\}$, where $u_i$ is the $i$-th seed picked by the greedy algorithm. Let $S_B^* = \{u_1, ... , u_B\}$. Due to the submodularity of $I(S)$ and the greedy algorithm, we have $I(S_B^*) \geq \frac{B}{K} I(S_K^*)$. Obviously $I(S_B^*) \geq I(S_B^*)$. Thus, we have $I(S_B^*) \geq \frac{B}{K} I(S_K^*)$. Moreover, when we take $C_B$, the corresponding configuration of $S_B^*$ as the initial value for coordinate descent iterations, we can find a configuration $C^*$ such that $UI(C^*) \geq UI(C_B) = I(S_B^*)$. Thus, Algorithm 1 is also $\alpha \frac{B}{K}$-optimal.

**Proof of Theorem 8**

Proof. By the definition of $UI(C)$, we have

$$\frac{UI(C)}{n} = \frac{1}{n} \sum_{S \subseteq V} Pr(S; V, C)I(S) = \sum_{S \subseteq V} Pr(S; V, C) \frac{I(S)}{n}$$

For a triggering model, by randomly generating a hyper-edge $h$ via a Monte Carlo simulation of the reverse propagation process, we have that $A(S, h)$ is an unbiased estimation of $\frac{I(S)}{n}$, where $A(S, h) = 1$ if $S \cap h \neq \emptyset$, and $A(S, h) = 0$ otherwise. So $E[A(S, h)] = \frac{I(S)}{n}$ and

$$\frac{UI(C)}{n} = \sum_{S \subseteq V} Pr(S; V, C)E[A(S, h)]$$
For the expectation $E[A(S,h)]$, the randomness is over the random hyper edge $h$. Denote by $Pr(h)$ the probability of generating a hyper-edge $h$. So $E[A(S,h)] = \sum_{h \subseteq V} Pr(h) A(S,h)$, and we have

$$\frac{UI(C)}{n} = \sum_{S \subseteq V} Pr(S;V,C) \sum_{h \subseteq V} Pr(h) A(S,h)$$

$$= \sum_{h \subseteq V} Pr(h) \sum_{S \subseteq V} Pr(S;V,C) A(S,h)$$

The sum $\sum_{S \subseteq V} Pr(S;V,C) A(S,h)$ can be regarded as the expectation of $A(S,h)$ when $h$ is fixed and $S$ is randomly generated according to $Pr(S;V,C)$. It is obvious that when $S \cap h = \emptyset$, $Pr(S;V,C) A(S,h) = 0$, and when $S \cap h \neq \emptyset$, $A(S,h) = 1$. So

$$\sum_{S \subseteq V} Pr(S;V,C) A(S,h) = \sum_{S \subseteq V, S \cap h \neq \emptyset} Pr(S;V,C)$$

The right-hand side is the probability that the randomly generated set $S$ has at least one node in $h$. Recall that $Pr(S;V,C) = \prod_{u \in S} p_u(c_u) \prod_{v \in V \setminus S} (1 - p_v(c_v))$, which means each $u$ belongs to $S$ independently. Thus, obviously,

$$\sum_{S \subseteq V} Pr(S;V,C) A(S,h) = \sum_{S \subseteq V, S \cap h \neq \emptyset} Pr(S;V,C)$$

$$= 1 - \prod_{u \in h} (1 - p_u(c_u))$$

Combining the above results, we have

$$\frac{UI(C)}{n} = \sum_{h \subseteq V} Pr(h) [1 - \prod_{u \in h} (1 - p_u(c_u))]$$

The right-hand side is the expectation of $1 - \prod_{u \in h} (1 - p_u(c_u))$, when $h$ is randomly generated according to $Pr(h)$. Thus,

$$\frac{UI(C)}{n} = E[1 - \prod_{u \in h} (1 - p_u(c_u))]$$

By the linearity of expectation, we have

$$E[n \times \sum_{h \in H} \frac{1 - \prod_{u \in h} (1 - p_u(c_u))}{m_H}] = UI(C).$$

Proof of Theorem 9

Proof. We prove the NP-hardness of the optimization problem in Eq. 15 by a reduction from the max $B$-set cover problem, which is known to be NP-hard. Consider an arbitrary instance of the max $B$-set cover problem, which consists of the set of all elements $E = \{e_1, \ldots, e_n\}$, and a collection of set $S = \{S_1, \ldots, S_m\}$, where $S_i \subseteq V$ for all $i$, and an integer $B$. The objective is to choose $B$ sets from $S$ such that the cardinality of the union of these $k$ sets is maximized. Denote by $OPT$ the optimal cardinality.
We reduce such an instance to an instance of Eq.\dagger. We first create a hyper-graph $H$ as follows. We set the node set $V$ as $\{1, \ldots, m\}$, and create $n$ hyper-edges $\{h_1, \ldots, h_n\}$. Note that a hyper-edge essentially is a set of nodes. For each $h_i$ and a node $u \in V$, we make $u \in h_i$ if $e_i \in S_u$. We also set the budget in Eq.\dagger as $B$, and for each $u \in V$, we set the seed probability function $p_u (c_u) = c_u$. The reduction is obviously in polynomial time.

Clearly, for any $S \subseteq V$, $\deg_H(S) = |\cap_{u \in S} S_u|$. Thus, we have $\max_{S \subseteq V:|S|=B} \deg_H(S) = OPT$. Now all we need to prove is that the optimality of Eq.\dagger is $\max_{S \subseteq V:|S|=B} \deg_H(S)$. This is equivalent to prove that Eq.\dagger in the instance constructed above has an optimal configuration $C$ whose entries are all integers (either 0 or 1).

The proof is pretty like the proof of Theorem 7. For any valid configuration $C$ such that $\sum_{u \in V} c_u = B$, if $C$ contains non-integer entries, it must contain at least 2 non-integer entries. We pick two non-integer entries $c_i$ and $c_j$, and fix all other entries. Suppose $c_i + c_j = B'$, then we can optimize over $c_i$ to further improve the objective $f_H(C)$. It is easy to verify that $f_H(C)$ is a convex function with respect to $c_i$ when satisfying $c_j = B' - c_i$, $0 \leq c_i \leq 1$ and $0 \leq c_j \leq 1$ (because the coefficient of the quadratic term is positive). Thus, similar to the construction in the proof of Theorem 7 after optimizing over $c_i$, either $c_i$ or $c_j$ becomes an integer and the objective $f_H(C)$ does not decrease. Therefore, there exist an integer configuration $C$ such that $f_H(C)$ is optimal to Eq.\dagger.

Proof of Theorem 11

Proof. We show that $\frac{I(H(C))}{m_H}$ can be represented by a convex combination of a finite number of unbiased estimations of influence spread of seed sets. $f_H(C) = \sum_{h \in H} [1 - \prod_{u \in h} (1 - p_u (c_u))]$. In the proof of Theorem 8 we demonstrate that

$$\sum_{S \subseteq V} Pr(S;V,C)A(S,h) = 1 - \prod_{u \in h} (1 - p_u (c_u))$$

where $A(S,h) = 1$ if $S \cap h \neq \emptyset$, and $A(S,h) = 0$ otherwise. Thus,

$$f_H(C) = \sum_{h \in H} [1 - \prod_{u \in h} (1 - p_u (c_u))]$$

$$= \sum_{h \in H} \sum_{S \subseteq V} Pr(S;V,C)A(S,h)$$

$$= \sum_{S \subseteq V} Pr(S;V,C) \sum_{h \in H} A(S,h)$$

Note that $\sum_{h \in H} A(S,h)$ is actually $\deg_H(S)$. So we have

$$\frac{f_H(C)}{m_H} = \sum_{S \subseteq V} Pr(S;V,C) \frac{\deg_H(S)}{m_H}$$

Since $Pr(S;V,C)$ is the probability of generating a seed set $S$ when the discount configuration is $C$, clearly $\frac{I(H(C))}{m_H}$ is a convex combination of $\frac{\deg_H(S)}{m_H}$ over all possible $S$. In addition, $\frac{UI(C)}{n} = \sum_{S \subseteq V} Pr(S;V,C) \frac{I(S)}{n}$ and $\frac{UI(C)}{n}$ is a convex combination of $\frac{I(S)}{n}$ with the same coefficients. Recall that $\frac{\deg_H(S)}{m_H}$ is an unbiased estimation of $\frac{I(S)}{n}$. If for all $S \subseteq V$, $|\frac{\deg_H(S)}{m_H} - \frac{I(S)}{n}| \leq \epsilon$, then it is guaranteed that $|\frac{I(H(C))}{m_H} - \frac{UI(C)}{n}| \leq \epsilon$. 38
We set $m_H = \frac{3(n \ln 2 + \ln \frac{2}{\epsilon^2})}{\epsilon^2}$. Applying the Chernoff bound [23], we obtain that for a seed set $S \subseteq V$,

$$Pr\{\left|\frac{\text{deg}_H(S)}{m_H} - \frac{I(S)}{n}\right| \geq \epsilon\} \leq \frac{\delta}{2^n}$$

Applying the union bound, we have

$$Pr\left\{\left|\frac{\text{deg}_H(S)}{m_H} - \frac{I(S)}{n}\right| \leq \epsilon, \forall S \subseteq V\right\} \geq 1 - \delta$$

Therefore, when $m_H = \frac{3(n \ln 2 + \ln \frac{2}{\epsilon^2})}{\epsilon^2}$,

$$Pr\left\{\left|\frac{f_H(C)}{m_H} - \frac{UI(C)}{n}\right| \leq \epsilon, \forall C \in \mathcal{C}\right\} \geq 1 - \delta.$$