Efficient Algorithms for Budgeted Profit Maximization with Theoretical Guarantees

Anonymous Author(s)

Abstract

Given a social network $G = (V, E)$, the unconstrained profit maximization problem aims to identify a subset $S \subseteq V$ that maximizes the net profit, defined as the expected influence spread $\Gamma(S)$ of set S minus the associated cost $c(S)$, i.e., $\Gamma(S) - c(S)$. However, this problem presupposes an unlimited budget, which is often impractical in real scenarios. Motivated by this, we formulate the new budgeted profit maximization (BPM) problem by adding a budget constraint. Unfortunately, addressing the BPM problem with a theoretical approximation guarantee remains open in the literature. In response, assuming $\Gamma(S)$ is known for any $S \subseteq V$, we propose an algorithm that guarantees returning a set S^o such that $\Gamma(S^o)-c(S^o) \geq \left(1-\frac{1}{e}\right)\frac{\Gamma(S^*)}{2}-\frac{c(S^*)}{2}$, where S^* denotes an optimal solution for BPM. Then, we develop a practical solution, which uses the reverse reachable set (RR-set) technique for influence estimation, without assuming knowledge of $\Gamma(S)$, while still maintaining a strong approximation guarantee.

Additionally, similar to existing RR-set-based solutions for influence cascade-related problems, our RR-set-based solution relies on generating a large number of random RR-sets to accurately estimate $\Gamma(S)$. However, the existing RR-set generation method suffers from high memory stall rates due to its irregular memory access patterns, leaving room for further efficiency improvement. Therefore, we propose a new RR-set generation method that utilizes batch execution and cache prefetching. When a memory access is required, instead of stalling while waiting for data, the CPU first issues an asynchronous prefetch request to load the target data into the cache, and then switches to processing the generation of other RR-sets within the same batch, effectively hiding memory access latency. This method can be seamlessly integrated into existing RR-set-based solutions to improve their efficiency. Finally, we conduct extensive experiments on real, large-scale datasets to demonstrate the effectiveness and efficiency of our proposed solutions.

ACM Reference Format:
Anonymous Author(s). 2018. Efficient Algorithms for Budgeted Profit Maximization with Theoretical Guarantees. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX).* ACM, New York, NY, USA, 16 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Given a social network G, an integer k, and a diffusion model, the *influence maximization (IM)* problem [31] seeks to identify k seed nodes that maximize the spread of influence across G. However, the classic IM framework overlooks the cost associated with seed selection, a factor that is critical in real-world commercial scenarios. To address this gap, researchers [36] introduced a variant of IM, known as *profit maximization (PM)*, which aims to optimize the net profit from marketing campaigns. The profit from a set of users S is quantified as the utility score minus the associated cost for these users. Typically, the utility score is measured by the expected influence spread $\Gamma(S)$ [36, 46] of set S under a certain cascade mode, and the cost function $c(S)$ is modular, expressed as $c(S) = \sum_{v\in S} c_v$, where c_v denotes the cost of selecting node v. Given its incorporation of the costs incurred in marketing activities, the PM problem has garnered considerable interest within the research community [26, 29, 45, 46].

Existing research on the PM problem often assumes an unrestricted total cost for the final seed set, with new nodes being added until no further positive profit is achievable. However, in practical scenarios, this approach may lead to significantly high marketing expenditure. While larger profits may eventually be realized, such unrestrained spending is not viable in a commercial setting. In particular, the returns from marketing campaigns materialize much later than the initial expenses. This time latency can strain a company's cash flow, particularly if total expenditures are substantial. Consequently, imposing a budget constraint is vital in real-world applications. Indeed online marketing agencies [19, 33, 49] often recommend that clients establish campaign budgets based on their overarching marketing goals, highlighting the importance of budget considerations in marketing strategies. Under a budget constraint, a company may fully utilize the budget for more profit or partially use it if the incremental benefits of recruiting additional influencers do not cover the recruiting costs. This approach ensures controlled spending, even if it may not always yield maximum profit compared to scenarios with an unlimited budget. However, it offers a more pragmatic and sensible strategy in real-world applications.

Motivated by the aforementioned limitation, we naturally introduce a new optimization problem that adds a budget constraint to the original unconstrained profit maximization problem. This is referred to as the *budgeted profit maximization (BPM)* problem. Although this extension is straightforward, designing an algorithm with a provable approximation guarantee for the BPM problem remains challenging. It is still an *open* problem in the literature. To explain, although the profit function (i.e., expected influence minus cost) is submodular, it may not consistently be non-negative, which is a crucial assumption in devising algorithms with provable guarantees for maximizing submodular functions [5, 10]. Furthermore, exactly computing the expected influence spread in polynomial time is computationally infeasible due to its #P-hard complexity [14, 16].

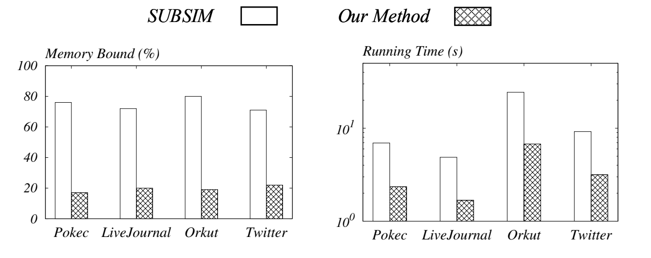
In response to these challenges, we first introduce a greedy-based algorithm, named GEAR, based on the assumption that a value oracle is available, which provides access to the expected influence spread value $\Gamma(S)$ in $O(1)$ time for any subset $S \subseteq V$. Specifically, in each iteration, GEAR selects the user with the highest positive marginal influence-to-cost ratio, and adds this user into the candidate set S_1 only if the budget is not exceeded (i.e., $c(S_1) \leq B$). Additionally, GEAR considers another singleton set S_2, which contains only one user that achieves the highest profit within the budget constraint. The algorithm then outputs the set, either S_1 or S_2, that yields the greater profit as the final solution. We conduct a comprehensive analysis of GEAR's approximation performance. It is shown that our solution GEAR returns a set S^o, ensuring

$$\Gamma(S^o) - c(S^o) \geq \frac{1}{2} \cdot \left((1 - \frac{1}{e})\Gamma(S^*) - c(S^*)\right),$$

where S^* denotes the optimal solution. To our knowledge, GEAR is the *first* algorithm that addresses the BPM problem with a provable approximation guarantee. Furthermore, since the assumption of having a value oracle for $\Gamma(S)$ is impractical, we utilize the reverse reachable sets (RR-sets) technique [9] to estimate $\Gamma(S)$. This method is widely used within the influence cascade community [29, 44, 47, 48]. We present a practical solution, RAP, with an approximation guarantee given by $\Gamma(S^o) - c(S^o) \geq \frac{(1-1/e)(1-\epsilon)\cdot\Gamma(S^*)-c(S^*)}{2}$, where $\epsilon \in (0, 1)$ denotes the error threshold parameter.

Furthermore, an RR-set-based solution (e.g., our RAP and other approaches [29, 44, 47, 48]) usually requires generating a substantial number of random RR-sets to accurately estimate influence, enabling the effective selection of truly influential seed nodes. The process of generating random RR-sets is a key factor in determining the overall efficiency of these solutions. Guo et al. [23] propose an efficient RR-set generation method, termed SUBSIM, which leverages geometric distribution sampling to skip unselected in-neighbors and directly *jump* to the next sampled in-neighbor. Although SUBSIM does improve the efficiency of RR-set generation, it, however, still suffers from high memory stall rates, leaving room for further improvement. To analyze SUBSIM's memory access performance, we use the Intel VTune Profiler, profiling the process of generating $2^{10} \times 10^3$ random RR-sets across four real-world datasets. The results are shown in Fig. 1(a), where the memory bound metric [27] measures the fraction of CPU cycles spent waiting due to demand load or store instructions. A higher value for this metric indicates more frequent stalls in the CPU pipeline slots caused by memory loads and stores. For instance, the memory bound value reaches as high as 80% on the dataset Orkut. This high stall rate can be attributed to the irregular data access pattern of SUBSIM, which is unfriendly to hardware cache prefetching, causing the CPU to frequently stall while waiting for the required data.

To address this weakness, we propose a solution called RISC, which combines batch execution with software cache prefetching. Specifically, when a memory access is required, instead of stalling while waiting for data, the CPU first issues an asynchronous prefetch request to load the target data into the cache, and then switches to processing the generation of other RR-sets within the same batch, effectively hiding memory access latency. Our RISC significantly reduces the memory bound metric (see Fig. 1(a)) and substantially improves the efficiency of random RR-set generation,



(a) Memory Bound metric (b) Running Time

Figure 1: Profiling analysis of random RR-set generation on four real datasets. The metric of memory bound indicates the fraction of CPU cycles spent waiting due to demand load or store instructions.

achieving a 3 ∼ 4x speedup in running time (see Fig. 1(b)). Notably, given that the RR-set technique is a fundamental tool in the literature, our new generation method can be directly applied to accelerate a plethora of existing solutions.

We summarize the main contributions of our paper as follows:

- We formulate the budgeted profit maximization (BPM) problem, which extends the original unconstrained profit maximization problem by considering a budget constraint. This adjustment offers a model that more accurately reflects real-world applications.
- Assuming the existence of an oracle that provides the expected influence spread for any subset $S \subseteq V$, we present a novel solution, GEAR, for the BPM problem. To the best of our knowledge, this is the *first* solution to address the BPM problem with a provable approximation guarantee.
- In the absence of an oracle for expected influence spread, we propose RAP, a practical approach that combines the RR-set technique with GEAR, while maintaining a theoretical guarantee.
- We propose RISC, which integrates batch execution with software cache prefetching to improve the efficiency of the random RR-set generation.
- Extensive experiments conducted on real-world datasets demonstrate the effectiveness and efficiency of our proposed solutions, highlighting their practical applicability.

2 Preliminary

We denote a directed graph as $G = (V, E)$, where V is the set of nodes, and E is the set of edges. Let $|V| = n$ and $|E| = m$. Each edge $e = (u, v)$ is associated with a probability, denoted as $p(e) \in [0, 1]$. For edge $e = (u, v)$, we say node u is an in-neighbor of node v and v is an out-neighbor of u. Let $N_{\text{in}}(u)$ be the set of u's in-neighbors, and $d_{\text{in}}(u)$ be the size of $N_{\text{in}}(u)$, i.e., $d_{\text{in}}(u) = |N_{\text{in}}(u)|$. Each node u is associated with a cost, denoted as c_u, representing the cost of selecting u as a seed. Let V_B denote the node set where each node has a cost no larger than the budget B, namely $V_B = \{u \mid c_u \leq B\}$.

2.1 Problem Definition

Diffusion model. In the literature, there exist two widely adopted diffusion models: *independent cascade (IC)* and *linear threshold (LT)* models [31]. In this paper, we focus on the IC model for ease of clarification. Our conclusion and algorithms can be extended to the LT model. In particular, under the IC model, given a node set $S \subseteq V$, the diffusion of influence is modeled as follows:

(i) at step $i = 0$, each node $u \in S$ is initially activated, leaving the remaining nodes inactive;

(ii) at each subsequent step $i > 0$, each node (e.g., u) that is newly activated in step $i-1$, has only *one* chance to activate its inactive out-neighbors (e.g., v), where the probability that u successfully activates node v is $p(u, v)$;

(iii) this diffusion terminates if no node can be activated anymore.

The *influence spread* of set S under the IC model is the total number of activated nodes when the diffusion process terminates.

Possible world and expected influence spread. Alternatively, we may interpret the influence diffusion by the concept of *live edge* [31]. In particular, for each edge $e \in E$ we independently flip a coin with head probability $p(e)$ to decide whether it is *live* or *blocked*. A *possible world*, denoted as ω, is an instance of flipping results of all edges. Then, under a possible world ω, the influence spread of set S, denoted as $\sigma_\omega(S)$, is the total number of nodes that are reachable from S via live edges. Let Ω denote the set of all possible worlds. Note that there are 2^m possible worlds, namely $|\Omega| = 2^m$. Next, given any seed set S, its expected influence spread, denoted as $\Gamma(S)$, is defined as $\Gamma(S) = \sum_{\omega \in \Omega} \sigma_\omega(S) \cdot p(\omega)$, where $p(\omega)$ is the probability that the possible world ω occurs. In addition, we use $\Delta(v \mid S)$ to denote the increase in expected influence spread when adding v to any seed set S, i.e., $\Delta(v \mid S) = \Gamma(S \cup \{v\}) - \Gamma(S)$.

Budgeted Profit Maximization. Given a seed set S, the profit earned by S can be defined as

$$\mathcal{P}(S) = \Gamma(S) - c(S) \tag{1}$$

where $\Gamma(S)$ is the expected influence spread of seed set S [36, 46] and $c(S)$ denotes the total cost of the node set S, defined as $c(S) = \sum_{u \in S} c_u$. The expected influence spread $\Gamma(S)$ represents the utility or benefit earned by the seed set S in a marketing campaign. Note that the function $\mathcal{P}(\cdot)$ may take a negative value. This can happen, for instance, when a node $u \in V$ requires a high cost while contributing insignificant influence spread.

Lemma 2.1. *[46] The function $\mathcal{P}(\cdot)$ is submodular yet non-monotonic.*

Next, we define the budgeted profit maximization problem (*BPM*).

Definition 2.2 (Budgeted Profit Maximization). Given a social network $G = (V, E)$ and a budget $B > 0$, the budgeted profit maximization problem aims to identify a seed set S with the largest profit and meanwhile $c(S) \leq B$,

$$S = \arg \max_{S' \colon c(S') \leq B} \mathcal{P}(S') = \arg \max_{S' \colon c(S') \leq B} \Gamma(S') - c(S'). \tag{2}$$

2.2 Profit Maximization

In this section, we revisit the existing solutions for the profit maximization problem in the literature. Previous research [25, 36, 45, 46, 52] has mainly addressed the profit maximization problem either without constraints or with a k-cardinality constraint (i.e., $|S| \leq k$). Our work is the *first* to explore the profit maximization problem under a budget constraint, which is sometimes referred to as a *knapsack* constraint. The existing solutions are listed as follows:

• **Simple Greedy** [36, 46]: This algorithm addresses the unconstrained profit maximization problem. It starts with an empty seed set $S = \emptyset$. Then, in each iteration, the algorithm selects a node u that yields the largest *marginal profit* from the remaining nodes in $V \backslash S$, defined as:

$$u = \arg \max_{v \in V \backslash S} \left[\mathcal{P}(S \cup \{v\}) - \mathcal{P}(S)\right] = \arg \max_{v \in V \backslash S} \left[\Delta(v \mid S) - c_v\right],$$

where u is then added to the set S. For simplicity in exposition, nodes that achieve the largest marginal gain in a greedy algorithm are referred to as *locally optimal nodes* in the rest of this paper. The process continues until no node in $V \backslash S$ has a positive marginal profit. The final set S is the output of this process. However, due to the submodular yet non-monotone nature of $\mathcal{P}(S)$, the Simple Greedy algorithm can perform arbitrarily worse than the optimal solution and lacks any bounded approximation factor [46].

• **ROI Greedy** [29]: Regarded as the leading solution for the unconstrained profit maximization problem, ROI Greedy similarly utilizes a greedy framework. It begins with an empty set S and iteratively adds locally optimal nodes into S. Unlike Simple Greedy, each iteration involves selecting the node u with the highest *ratio of marginal influence-to-cost*, i.e., $\frac{\Delta(u|S)}{c_u}$. If this locally optimal node u can produce a positive marginal profit, namely, $\Delta(u \mid S) - c_u > 0$, add it to the set S, otherwise the algorithm is terminated. Assuming S^* represents the solution with optimal profit, Jin et al. demonstrate that ROI Greedy ensures:

$$\Gamma(S) - c(S) \geq \Gamma(S^*) - c(S^*) - \ln\left(\frac{\Gamma(S^*)}{c(S^*)}\right) \cdot c(S^*). \tag{3}$$

• **Distorted Greedy** [25, 30]: This algorithm is designed for the profit maximization problem under a k-cardinality constraint (i.e., $|S| \leq k$). It generates a set S that satisfies:

$$\Gamma(S) - c(S) \geq \left(1 - \frac{1}{e}\right)\Gamma(S^*) - c(S^*),$$

where S^* is the optimally profitable solution. The algorithm starts with an empty set S, and in each iteration i (for $i = 1, 2, 3, \ldots, k$), it identifies the node u that maximizes the marginal *distorted profit*, defined as:

$$u = \arg \max_{v \in V} \left(1 - \frac{1}{k}\right)^{k-i} \cdot \Delta(u \mid S) - c_u.$$

The node u is added to S only if its marginal distorted profit is positive. By setting $k = n$, Distorted Greedy can also be adapted to address the unconstrained profit maximization problem.

Non-monotone Submodular Maximization. The literature includes studies [5, 6, 10, 38] that aim to maximize a non-monotone submodular function. To derive an approximation guarantee, these studies commonly assume that the function takes non-negative values for any subset $S \subseteq V$. However, this assumption does not hold in the context of profit maximization.

More recently, Amanatidis et al. [5, 6] introduced a randomized solution, **Sample Greedy**, for the non-monotone submodular maximization problem under a knapsack constraint, which expectedly achieves a $(3 + 2\sqrt{2})$-approximation guarantee. Sample Greedy begins by selecting a random subset V' of V, sampling each node independently with a probability of $p = \sqrt{2} - 1$. It then applies Wolsey's algorithm [51] to the subset V'. As the final solution provided by Sample Greedy is derived from the randomized subset V', it offers only an expected approximation guarantee. Moreover, this theoretical conclusion is infeasible for our budgeted profit maximization problem, as it requires the prerequisite that $\mathcal{P}(S) \geq 0$ for any subset $S \subseteq V$.

2.3 Reverse Reachable Set

The task of exactly computing the expected influence $\Gamma(S)$ has been recognized as #P-hard [14, 16]. To address this problem, the concept of a reverse reachable set (RR-set) was first proposed by Borgs et al. [9] and has been widely applied to efficiently estimate the expected influence spread in the influence cascade domain [23, 47, 48]. A *random* RR-set is constructed as follows:

(i) randomly sample a node $v \in V$;
(ii) starting from node v, perform a *stochastic* Breadth-First Search (BFS) on the transpose graph G^T (i.e., the direction of each edge in G is reversed);

The term "stochastic" implies that an edge e adjacent to the current frontier node is traversed with probability $p(e)$, rather than in a deterministic manner. The set of all discovered nodes forms an RR-set, denoted by R, with node v being the *target node* of R. An RR-set can be understood as the set of nodes that reaches the target node v under the corresponding possible world.

Lemma 2.3. *[9] Let S be a seed set and R be a random RR-set, then*

$$\Gamma(S) = n \cdot \mathbb{P}[S \cap R \neq \emptyset].$$

A seed set S is said to *cover* an RR-set R if their intersection is non-empty, i.e., $S \cap R \neq \emptyset$. The coverage of S in a collection $\mathcal{R}$ of RR-sets is defined as the number of RR-sets covered by S, denoted as $\Lambda_{\mathcal{R}}(S)$. From Lemma 2.3, $n \cdot \frac{\Lambda_{\mathcal{R}}(S)}{|\mathcal{R}|}$ serves as an unbiased estimator of the influence of the seed set S. By concentration bounds (e.g., the Chernoff Bound), we know that the larger the size of $\mathcal{R}$, the more accurate the estimation becomes. Therefore, RR-set-based solutions [29, 44, 47, 48] typically sample a collection of RR sets, and based on this collection, select a seed set S with a coverage as large as possible. If this seed set satisfies the approximation requirement, it is returned as the final solution. In these RR-set-based solutions, the RR-set serves as a common foundation. Thus, studying how to efficiently generate RR-sets becomes exceptionally valuable.

Recently, Guo et al. [23] presented an efficient RR-set generation method, SUBSIM, for the IC model. The pseudo-code of SUBSIM for the WC model (a special case of the IC model, where $p(u,v) = \frac{1}{d_{\text{in}}(v)}$) is presented in Alg. 1. In particular, when a node is newly activated, it utilizes geometric distribution sampling to skip in-neighbors that are not successfully sampled (see Lines 9–19), avoiding the need to scan the entire neighbor set and thus improving efficiency. They further extend Alg. 1 by adding rejection operations to handle the general IC model, where the probability distribution is skewed [24]. However, as mentioned in the Introduction, although SUBSIM improves sampling efficiency, its irregular memory access pattern is unfriendly to the CPU's automatic data prefetching, leaving room for further efficiency improvement.

Algorithm 1: SUBSIM[23]

```
Input: Input Graph $G = (V, E)$
Output: A random RR-set $R$
1  Randomly sample a node $v \in V$;
2  $R \leftarrow \{v\}$;
3  Add $v$ to queue $Q$ and mark $v$ as activated;
4  while $Q$ is not empty do
5      $u \leftarrow$ pop the top element of $Q$;        // BFS frontier
6      Let $u[i]$ be the $i$-th in-neighbor of $u$, where $i = 1, 2, \ldots$;
7      $p \leftarrow \frac{1}{d_{\text{in}}(u)}$;
8      $i \leftarrow 0,\ \mathcal{I} \leftarrow \emptyset$;
9      while $i \leq d_{in}(u)$ do
10         $i$ += $\lceil \log(rand())/\log(1-p) \rceil$;
11         $\mathcal{I} \leftarrow \mathcal{I} \cup \{i\}$;
12     end
13     for each $i \in \mathcal{I}$ do
14         $w \leftarrow u[i]$;
15         if $w$ is not activated then
16             Add $w$ to $R$;
17             Add $w$ to queue $Q$ and mark $w$ as activated;
18         end
19     end
20 end
21 return $R$;
```

3 Solutions for BPM

In this section, we present our algorithm for the BPM problem. To facilitate our analysis, we assume the existence of a value oracle that can return the value of $\Gamma(S)$ for any subset $S \subseteq V$ in $O(1)$ time. This assumption is commonly made in algorithm design [40, 50] to streamline the development of efficient algorithms. With this oracle, we propose our solution GEAR[1] in Section 3.1, which offers an approximation guarantee. Then, in Section 3.2, we further address how to handle the BPM problem in the absence of such an oracle.

3.1 GEAR Algorithm

At a high level, GEAR mainly adopts the cost-effective greedy strategy, analogous to ROI Greedy. The primary distinctions between ROI Greedy and GEAR are two-fold: (i) our method may reject certain locally optimal nodes due to budget constraints; (ii) the final solution returned is the one with the larger profit, chosen between the seed set S^+, which is obtained via the cost-effective rule, and the singleton set $\{v^*\}$. Here, the node v^* represents the node with the largest profit among those in the set V_B.

Alg. 2 presents the pseudocode of GEAR. Initially, the set S^+ is initialized as an empty set, and the flag variable *nodeRejected* is set to be *false*. Since including any node with a cost greater than B is not feasible, the candidate set V' is defined as V_B, where $V_B = \{v \mid c_v \leq B\}$. The algorithm then enters a while loop, which iteratively selects nodes based on the cost-effective rule. In each iteration, it identifies the locally optimal node with the highest marginal influence per unit cost, denoted as $\frac{\Delta(v|S^+)}{c_v}$, with respect to the current S^+. Let u be this locally optimal node. The loop terminates if u does not yield a positive marginal profit, namely, $\Delta(u \mid S^+) - c_u \leq 0$. Additionally, if the inclusion of u into S^+ still satisfies the budget constraint, the algorithm adds node u to S^+. Conversely, if the budget constraint is

[1]Cost-effective Greedy with approximation guarantees

Algorithm 2: GEAR

```
Input: Graph $G = (V, E)$, budget $B$.
Output: A seed set $S^o$.
1  $S^+ \leftarrow \emptyset$, $V' \leftarrow V_B$;
2  $nodeRejected \leftarrow false$;
3  while $V' \neq \emptyset$ do
4      $u \leftarrow \arg\max_{v \in V'} \frac{\Delta(v|S^+)}{c_v}$;
5      if $\Delta(u \mid S^+) - c_u \leq 0$ then
6          break;
7      end
8      if $c(S^+) + c_u \leq B$ then            // budget constraint
9          $S^+ \leftarrow S^+ \cup \{u\}$;
10     else
11         $nodeRejected \leftarrow true$;
12     end
13     $V' \leftarrow V' \backslash \{u\}$;
14 end
15 $v^* \leftarrow \arg\max_{v \in V_B} \Gamma(\{v\}) - c(v)$;
16 $S^o \leftarrow \arg\max\{\Gamma(S^+) - c(S^+), \Gamma(\{v^*\}) - c_{v^*}\}$;
17 return $S^o$;
```

violated, it rejects u and sets *nodeRejected* to *true*. This flag indicates that at least one locally optimal node was rejected due to budget constraints during the loop. After exiting the while loop, the seed set S^+ is finalized. Subsequently, another seed set $\{v^*\}$ is constructed, where the singleton node v^* represents the node with the highest profit among all nodes in V_B. The algorithm then returns the set with the larger profit, either S^+ or $\{v^*\}$. The *nodeRejected* variable, while redundant for Alg. 2, is included in the pseudocode to facilitate analyzing its approximation performance. It can be removed when implementing the algorithm GEAR.

Given the budget constraint, it is intuitively logical to exclude nodes that would exceed the budget. However, this decision substantially complicates the analysis of its approximation performance. Despite these challenges, we have successfully derived the following theorem to demonstrate the superior profit performance of GEAR compared to any other set $S \subseteq V$ with $c(S) \leq B$.

Theorem 3.1. *Give budget $B > 0$, the algorithm GEAR returns a seed set S^o which achieves the following approximation guarantee:*

$$\Gamma(S^o) - c(S^o) \geq \frac{1}{2} \cdot \left(\max_{S: c(S) \leq B} (1 - \frac{1}{e})\Gamma(S) - c(S) \right) \tag{4}$$

To prove Theorem 3.1, we thoroughly analyze all possible outcomes of GEAR, and show Inequality (4) always holds. Please refer to Section 3.3 for proof details. Then, according to Theorem 3.1 we can give an immediate corollary as follows.

Corollary 3.2. *GEAR returns a solution S^o such that*

$$\Gamma(S^o) - c(S^o) \geq \frac{1}{2} \cdot \left((1 - \frac{1}{e})\Gamma(S^*) - c(S^*) \right)$$

where $S^* = \arg\max_{S: c(S) \leq B} \Gamma(S) - c(S)$.

Remark. Let $f(S) = g(S) - l(S)$, where $g(S)$ is a non-negative monotone submodular function, and $l(S)$ is a non-negative modular

Algorithm 3: RAP

```
Input: $G = (V, E)$, budget $B$, parameter $\epsilon$ and $\delta$.
Output: A seed set $S^o$.
1  $\mathcal{R}_1 \leftarrow \emptyset$, $\mathcal{R}_2 \leftarrow \emptyset$;
2  $\omega_1 \leftarrow \log n$, $i \leftarrow 1$, $\alpha \leftarrow (1 - 1/e)/2$;
3  do
4      Insert new random RR-sets into $\mathcal{R}_1$ and $\mathcal{R}_2$ respectively
         until $|\mathcal{R}_1| = |\mathcal{R}_2| = \omega_i$;
5      $S^o \leftarrow$ SeedSelection($\mathcal{R}_1$);
6      if VerifySeedSet($S^o, \mathcal{R}_1, \mathcal{R}_2$) then
7          break;
8      end
9      $\omega_{i+1} \leftarrow 2 \cdot \omega_i$, $i \leftarrow i + 1$;
10 while $|\mathcal{R}_1| \leq \frac{(8+2\alpha\epsilon)(1+\epsilon_1)n(\ln(6/\delta)+n\ln 2)}{\epsilon^2 \cdot \alpha^2 \cdot \max\{1, \Gamma_2(S^o) - (1+\epsilon_1)c(S^o)\}}$;
11 return $S^o$;
```

function. The task of maximizing the function $f(S)$ is termed *Regularized Submodular Maximization* (RSM) in the literature, which has garnered significant research attention [30, 35]. In proving Theorem 3.1, we rely solely on the non-negative monotone submodular nature of $\Gamma(S)$ and the non-negative modular nature of $c(S)$. Consequently, GEAR can be directly applied to solve the constrained RSM problem that seeks to maximize $f(S)$ subject to $l(S) \leq B$, offering a strong approximation guarantee. To the best of our knowledge, GEAR is the *first* solution with a provable approximation guarantee for the RSM problem subject to a knapsack constraint.

3.2 RAP Algorithm

In real-world scenarios, a value oracle model, where the expected influence spread $\Gamma(S)$ can be accessed in $O(1)$ time, does not exist. Motivated by this limitation, in this section, we develop a practical solution RAP which combines GEAR with the RR-set to address the BPM problem in the absence of a value oracle for $\Gamma(S)$. From a high-level perspective, we generate random RR-sets to provide influence estimation, and subsequently employ GEAR to identify a candidate set S^o. Upon generating a sufficient number of random RR-set instances, we can ensure the accuracy of estimating $\Gamma(S)$, thereby guaranteeing the quality of S^o.

Our solution, named *RAP*[2], mainly adopts the *SelectAndVerify* framework [22, 29, 44]. Alg. 3 presents the pseudocode of RAP. In particular, it maintains two collections of RR-sets, denoted as $\mathcal{R}_1$ and $\mathcal{R}_2$, where $\mathcal{R}_1$ is used for seed selection and $\mathcal{R}_2$ is used for seed quality verification. In the while loop, it first calls Procedure SeedSelection, which employs GEAR to obtain a candidate seed set S^o. Subsequently, it verifies whether S^o meets the desired approximation guarantee via Procedure VerifySeedSet. If the requirement is met, the algorithm stops immediately; otherwise, it doubles the size of $\mathcal{R}_1$ and $\mathcal{R}_2$. This process repeats until a qualified seed set S^o is identified, or $|\mathcal{R}_1|$ reaches the predefined threshold (Line 10).

SeedSelection. Specifically, the candidate seed set S^o is obtained by invoking the algorithm GEAR, where the expected influence function $\Gamma(S)$ is replaced by the estimator $\frac{\Lambda_{\mathcal{R}_1}(S)}{|\mathcal{R}_1|} \cdot |V|$. When the size

[2] RR-set based Algorithm for Budgeted Profit Maximization.

```
Procedure SeedSelection($\mathcal{R}_1$)
1 Let $\Lambda_{\mathcal{R}_1}(S)$ be the number of RR-sets covered by $S$ in $\mathcal{R}_1$;
2 $\Gamma(S) \leftarrow \frac{\Lambda_{\mathcal{R}_1}(S)}{\omega_i} \cdot |V|$;            // influence estimator
3 $S^o \leftarrow$ invoke Algorithm GEAR;
4 return $S^o$;
```

of $\mathcal{R}_1$ is sufficiently large, the estimator converges to its expectation $\Gamma(S)$ according to Lemma 2.3. Further, Theorem 3.1 guarantees that the selected S^o probably gives a superior profit performance.

VerifySeedSet. From the generation of $\mathcal{R}_1$ and $\mathcal{R}_2$, it is evident that these two sets are independent. Consequently, $\mathcal{R}_2$ can be utilized to verify whether S^o meets the desired approximation guarantee. At a high level, the variable ϵ_1 quantifies the error deviation between the estimated value $\frac{\Lambda_{\mathcal{R}_2}(S^o)}{|\mathcal{R}_2|} \cdot |V|$ and its true value $\Gamma(S^o)$. Similarly, the variable ϵ_2 quantifies the error deviation between the estimated value $\frac{\Lambda_{\mathcal{R}_1}(S^*)}{|\mathcal{R}_1|} \cdot |V|$ and the true value $\Gamma(S^*)$, where S^* represents the optimal solution. When both error variables are bounded by the specified error parameter ϵ, we can ensure that the obtained seed set S^o satisfies the required approximation.

Theorem 3.3. *Given $\epsilon \in (0,1)$ and budget B, the algorithm RAP returns a seed set S^o which with probability at least $1-\delta$, satisfies*

$$\Gamma(S^o) - c(S^o) \geq \frac{1}{2}\left((1-1/e)(1-\epsilon)\cdot\Gamma(S^*) - c(S^*)\right),$$

where S^* has the largest profit under the constraint of budget B.

Lemma 3.4. *The algorithm RAP runs in $O\left(\frac{n\cdot\tau}{\epsilon^2}\cdot\max\{\ln\delta^{-1}, n\}\right)$ time, where τ is the expected cost to sample a random RR-set.*

3.3 Theoretical Analysis

In this section, we present the proof of Theorem 3.1, Theorem 3.3 and Lemma 3.4.

Proof of Theorem 3.1. Let $S^\#$ denote the optimal seed set that maximizes the *distorted* profit, specifically defined as:

$$S^\# = \arg\max_{S: c(S)\leq B}\left(1-\frac{1}{e}\right)\Gamma(S) - c(S).$$

We assume the maximum distorted profit, $\left(1-\frac{1}{e}\right)\Gamma(S^\#) - c(S^\#)$, is non-negative. If this is not the case, as the final solution, we return S^o if $\mathcal{P}(S^o) \geq 0$ or an empty set otherwise, and Inequality (4) is immediately satisfied.

Let $U = \{u_1, u_2, \ldots, u_i, \ldots\}$ be the ordered set where u_i is the locally optimal node in the i-th iteration, namely, node u_i having the largest marginal influence per unit cost (see Alg. 2 Line 4). When the while loop terminates, we consider the following two cases:

Case 1. The variable *nodeRejected* remains *false*. In this case, no candidate node $u_i \in U$ is rejected due to the violation of the budget constraint. This indicates that the budget B is sufficiently large to include all nodes in U, i.e., $S^+ = U$. Thus, the algorithm GEAR yields the same solution as in unconstrained profit maximization

```
Procedure VerifySeedSet($S^o, \mathcal{R}_1, \mathcal{R}_2$)
1 $a \leftarrow \ln(6\cdot i^2/\delta)$ ;
2 $\Gamma_1(S^o) \leftarrow \frac{\Lambda_{\mathcal{R}_1}(S)}{\omega_i}\cdot|V|$,   $\Gamma_2(S^o) \leftarrow \frac{\Lambda_{\mathcal{R}_2}(S)}{\omega_i}\cdot|V|$;
3 $\epsilon_1 \leftarrow \frac{\sqrt{1+8\Lambda_{\mathcal{R}_2}(S^o)/a}-3}{2[(\Lambda_{\mathcal{R}_2}(S^o)/a)-1]}$;
4 $\epsilon_2 \leftarrow \sqrt{\frac{2(1+\epsilon_1)a\cdot n}{(\Gamma_2(S^o)-(1+\epsilon_1)c(S^o))\cdot\omega_i}}$;
5 $t \leftarrow (\Gamma_1(S^o)-c(S^o))/(\Gamma_2(S^o)-c(S^o))$;
6 $\mathcal{E}_0 \leftarrow (\epsilon_1 > 0)$ and $(\epsilon_2 > 0)$;
7 $\mathcal{E}_1 \leftarrow (t > 1)$ and $((1-1/t)+\epsilon_1+\epsilon_2 \leq \epsilon)$;
8 $\mathcal{E}_2 \leftarrow (0 < t \leq 1)$ and $(\epsilon_1+\epsilon_2 \leq \epsilon)$;
9 if ($\mathcal{E}_0$ and $\mathcal{E}_1$) or ($\mathcal{E}_0$ and $\mathcal{E}_2$) then
10     return true;
11 end
12 return false;
```

(see [29]). Consequently, according to the analysis in [29], it holds:

$$\Gamma(S^+) - c(S^+) \geq \Gamma(S^\#) - c(S^\#) - \ln\left(\frac{\Gamma(S^\#)}{c(S^\#)}\right)\cdot c(S^\#).$$

Next, observe that:

$$\left(\Gamma(S^\#) - c(S^\#) - \ln\left(\frac{\Gamma(S^\#)}{c(S^\#)}\right)\cdot c(S^\#)\right) - \left((1-\frac{1}{e})\Gamma(S^\#) - c(S^\#)\right)$$
$$= \left(\frac{1}{e}\cdot\frac{\Gamma(S^\#)}{c(S^\#)} - \ln\left(\frac{\Gamma(S^\#)}{c(S^\#)}\right)\right)\cdot c(S^\#).$$

Let $g(x) = \frac{1}{e}x - \ln x$. Differentiating the function $g(x)$ gives $g'(x) = \frac{1}{e} - \frac{1}{x}$, indicating that $g(x)$ attains its minimum at $x = e$ with $g(e) = 0$. Thus, Inequality (4) holds:

$$\Gamma(S^+) - c(S^+) \geq (1-\frac{1}{e})\Gamma(S^\#) - c(S^\#)$$
$$\geq \frac{1}{2}(1-\frac{1}{e})\Gamma(S^\#) - \frac{1}{2}c(S^\#).$$

Case 2. The variable *nodeRejected* is set to be *true*, indicating that we have rejected some locally optimal nodes as adding these nodes will violate the budget limit. Define $T_1 = S^\#\backslash S^+$ and $T_2 = S^+\backslash S^\#$, where "\" denotes the set subtraction operation. We then consider two subcases depending on whether T_2 and T_1 are empty.

Case 2.1. In the case of $T_2 = \emptyset$, this implies $S^+ \subseteq S^\#$. If $T_1 = \emptyset$, i.e., $S^+ = S^\#$, the inequality is directly satisfied. Otherwise, there exists at least one node $v \in S^\#$ and $v \notin S^+$. For such a node v, since $S^+ \cup \{v\} \subseteq S^\#$, it follows that $c(S^+) + c_v \leq c(S^\#) \leq B$, suggesting that node v should be included in S^+ if $\Gamma(v \mid S^+) - c_v > 0$. Therefore, we deduce that $\Gamma(v \mid S^+) - c_v \leq 0$. It follows:

$$\Gamma(S^+) - c(S^+) \geq \Gamma(S^+) - c(S^+) + \sum_{v\in T_1}\left(\Gamma(v \mid S^+) - c_v\right)$$
$$\geq \Gamma(S^\#) - c(S^\#) \geq (1-\frac{1}{e})\Gamma(S^\#) - c(S^\#).$$

where the second inequality comes from the fact that $\Gamma(S^+) + \sum_{v\in T_1}\Gamma(v \mid S^+) \geq \Gamma(S^\#)$ by submodularity. Thus, the inequality (4) follows in this case.

Case 2.2. If $T_2 \neq \emptyset$ and $T_1 = \emptyset$, then it follows that $S^\# \subset S^+$. To demonstrate the approximation guarantee, consider the unconstrained profit problem Q with the universe of nodes $V = S^+$, and the expected influence function $\Gamma(\cdot)$ where the marginal influence values are the same as in our problem. Then, run the algorithm ROI Greedy [29] for problem Q to obtain the solution S^+_Q. We can infer that $S^+_Q = S^+$, since in each iteration, both algorithms select the node with the highest marginal influence per unit cost (excluding rejection iterations in our algorithm GEAR). Besides, the seed set $S^\#_Q$ with the highest distorted profit for problem Q is the same as $S^\#$. If there were another set S'_Q such that $(1-\frac{1}{e})\Gamma(S'_Q) - c(S'_Q) > (1-\frac{1}{e})\Gamma(S^\#) - c(S^\#)$, and given $S'_Q \subseteq S^+$, it leads to $c(S'_Q) \leq c(S^+) \leq B$. Hence, S'_Q is a feasible solution for our budgeted problem and yields a higher distorted profit than $S^\#$, contradicting the optimality of $S^\#$. Therefore, $S^\#_Q = S^\#$. Following the discussions in Case 1, we confirm that Inequality (4) holds.

Case 2.3. In this scenario, $T_2 \neq \emptyset$ and $T_1 \neq \emptyset$. The condition $T_1 \neq \emptyset$ implies the existence of nodes $u \in S^\#$ that are not included in S^+. This situation can lead to two possibilities: (a) there exists a node $u \in T_1$ that is rejected due to the violation of budget constraint B (i.e., it fails to meet the condition in Alg. 2 Line 8); (b) all nodes in T_1 are not added to S^+ because the while loop is terminated due to the non-positive marginal profit condition (see Alg. 2 Line 5). We will explore these two sub-cases in the following discussion.

Case 2.3(a). There exists a node $u \in T_1$ that is rejected due to the violation of the budget constraint B. Recall that $U = \{u_1, u_2, \ldots, u_i, \ldots\}$ denotes the ordered set where u_i is the i-th locally optimal node. Let u_{j_i} be the i-th node added to S^+. Define $u_{j_{k+1}}$ as the first node such that $u_{j_{k+1}} \in S^\#$ but is not added to S^+. Let S_i be the set of the first i nodes added to S^+, namely, $S_i = \{u_{j_1}, u_{j_2}, \ldots, u_{j_i}\}$, for $i = 1, 2, 3, \ldots, k+1$.

Define k' as the number satisfying $c(S_{k'}) < c(S^\#) \leq c(S_{k'+1})$. Clearly, $k' \leq k$. Let t be the fraction defined by $\frac{c(S^\#)-c(S_{k'})}{c_{u_{j_{k'+1}}}}$. We have $t \in (0, 1]$. Then, we derive that:

$$\Gamma(S_{k'}) + t \cdot \Delta(u_{j_{k'+1}} \mid S_{k'}) \geq \left(1 - \frac{1}{e}\right)\Gamma(S^\#). \tag{5}$$

The derivation of Inequality (5) will be presented shortly.

Let $v^* = \arg\max_{v \in V_B} (\Gamma(v) - c_v)$. Additionally, for $i \leq k$, as v_{j_i} has a positive marginal profit (see Line 5), it holds that

$$\Gamma(S_{i+1}) - c(S_{i+1}) > \Gamma(S_i) - c(S_i). \tag{6}$$

With Inequality(5) and (6), we obtain

$$\begin{aligned}
&2 \cdot \left(\Gamma(S^o) - c(S^o)\right) = 2 \cdot \max\left\{\Gamma(S^+) - c(S^+),\, \Gamma(\{v^*\}) - c_{v^*}\right\} \\
&\geq 2 \cdot \max\left\{\Gamma(S_k) - c(S_k),\, \Gamma(\{v^*\}) - c_{v^*}\right\} \\
&\geq \Gamma(S_k) - c(S_k) + \Gamma(\{v^*\}) - c_{v^*} \\
&\geq \Gamma(S_k) - c(S_k) + \Gamma(\{v_{j_{k+1}}\}) - c_{v_{j_{k+1}}} \\
&\geq \Gamma(S_{k+1}) - c(S_{k+1}) \geq \Gamma(S_{k'+1}) - c(S_{k'+1}) \\
&\geq \Gamma(S_{k'}) - c(S_{k'}) + t \cdot \left(\Delta(v_{j_{k'+1}} \mid S_{k'}) - c_{v_{j_{k'+1}}}\right) \\
&\geq \left(1 - \frac{1}{e}\right)\Gamma(S^\#) - c(S^\#).
\end{aligned}$$

where the fourth inequality is due to the submodularity property $\Gamma(v_{j_{k+1}}) \geq \Gamma(v_{j_{k+1}} \mid S_k)$, and the sixth inequality follows from $\Delta(v_{j_{k'+1}} \mid S_{k'}) - c_{v_{j_{k'+1}}} > 0$. Thus, we reach Inequality(4).

Case 2.3(b). Since the non-positive profit condition stops the while loop, all nodes $u \in T_1$ are not added to S^+. Let $U_d = U \backslash S^+$. Thus, U_d is the set of locally optimal nodes rejected due to the budget constraint. Clearly, in this case, $S^\# \cap U_d = \emptyset$. To establish the approximation guarantee in this case, consider the unconstrained profit problem Q with the universe of nodes $V = V \backslash U_d$ and the same influence spread function as our budgeted problem. The seed set with the largest distorted profit for Problem Q is still $S^\#$. The solution returned by ROI Greedy [29] is also the same as S^+. Therefore, akin to Case 2.2, we can immediately conclude that the approximation guarantee of Inequality(4) is satisfied.

Based on the above discussions, we have considered all possible outputs for Alg. 2, and thus, the proof of Theorem 3.1 is complete.

Proof of Inequality (5). As initially $V' = V_B$, we know that at the beginning of the j_i-th iteration, $V' = V_B \backslash \{u_1, u_2, \ldots, u_{j_i - 1}\}$ where u_l is the identified locally optimal node in the l-th iteration. We first show for $i = 1, 2, \ldots, k+1$,

$$\begin{aligned}
\frac{\Delta(v_{j_i} \mid S_{i-1})}{c_{v_{j_i}}} &= \max_{v \in V'} \frac{\Delta(v \mid S_{i-1})}{c_v} \\
&\geq \max_{v \in S^\# \backslash S_i} \frac{\Delta(v \mid S_{i-1})}{c_v} \geq \frac{\sum_{v \in S^\#} \Delta(v \mid S_{i-1})}{\sum_{v \in S^\#} c_v} \\
&\geq \frac{\Gamma(S^\# \cup S_{i-1}) - \Gamma(S_{i-1})}{c(S^\#)} \geq \frac{\Gamma(S^\#) - \Gamma(S_{i-1})}{c(S^\#)}.
\end{aligned}$$

In the following derivation, for ease of illustration, we use c_i to denote $c_{v_{j_i}}$. We can rearrange the above inequality as:

$$\Gamma(S_i) \geq \frac{c_i}{c(S^\#)}\Gamma(S^\#) + \left(1 - \frac{c_i}{c(S^\#)}\right)\Gamma(S_{i-1}). \tag{7}$$

Recall that k' satisfies that $c(S_{k'}) < c(S^\#) \leq c(S_{k'+1})$, and $t = \frac{c(S^\#)-c(S_{k'})}{c_{k'+1}}$. Therefore, we have $c_i \leq c(S^\#)$ for $i = 1, \ldots, k'$ and $t \cdot c_{k'+1} \leq c(S^\#)$. Then we use induction to show: for $i = 1, 2, \ldots, k'$,

$$\Gamma(S_i) \geq \left(1 - \prod_{l=1}^{i}\left(1 - \frac{c_l}{c(S^\#)}\right)\right)\Gamma(S^\#). \tag{8}$$

For $i = 1$, from Inequality (7), we have $\Gamma(S_1) \geq \frac{c_1}{c(S^\#)}\Gamma(S^\#)$, where $\Gamma(S_0) = 0$. Assume that for $i = j$, it holds that

$$\Gamma(S_j) \geq \left(1 - \prod_{l=1}^{j}\left(1 - \frac{c_l}{c(S^\#)}\right)\right)\Gamma(S^\#).$$

Then for $i = j+1$, with Inequality (7), we have

$$\begin{aligned}
\Gamma(S_{j+1}) &\geq \frac{c_{j+1}}{c(S^\#)}\Gamma(S^\#) + \left(1 - \frac{c_{j+1}}{c(S^\#)}\right)\Gamma(S_j) \\
&\geq \frac{c_{j+1}}{c(S^\#)}\Gamma(S^\#) + \left(1 - \frac{c_{j+1}}{c(S^\#)}\right) \cdot \left(1 - \prod_{l=1}^{j}\left(1 - \frac{c_l}{c(S^\#)}\right)\right)\Gamma(S^\#) \\
&\geq \left(1 - \prod_{l=1}^{j+1}\left(1 - \frac{c_l}{c(S^\#)}\right)\right)\Gamma(S^\#).
\end{aligned}$$

Therefore, we complete the proof of Inequality (8).

We will use The Arithmetic-Geometric Mean inequality: for $x_1, x_2, \dots, x_n, y \in \mathbb{R}^+$, and $x_i \le y$ for $i \le y$, it holds that

$$\prod_{i=1}^{n}\left(1-\frac{x_i}{y}\right) \le \left(\frac{\sum_{i=1}^{n}(1-x_i/y)}{n}\right)^n = \left(1-\frac{\sum_{i=1}^{n}x_i}{ny}\right)^n. \tag{9}$$

Next, with Inequality (7) and (9), we obtain

$$\begin{aligned}
&\Gamma(S_{k'}) + t\cdot\Delta(v_{j_{k'+1}} \mid S_{k'})\\
\ge&\Gamma(S_{k'}) + \frac{t\cdot c_{k'+1}}{c(S^\#)}\cdot\left(\Gamma(S^\#)-\Gamma(S_{k'})\right)\\
\ge&\frac{t\cdot c_{k'+1}}{c(S^\#)}\Gamma(S^\#) + \left(1-\frac{t\cdot c_{k'+1}}{c(S^\#)}\right)\cdot\Gamma(S_{k'})\\
\ge&\frac{t\cdot c_{k'+1}}{c(S^\#)}\Gamma(S^\#) + \left(1-\frac{t\cdot c_{k'+1}}{c(S^\#)}\right)\left[1-\prod_{l=1}^{k'}\left(1-\frac{c_l}{c(S^\#)}\right)\right]\Gamma(S^\#)\\
\ge&\left[1-\left(1-\frac{t\cdot c_{k'+1}}{c(S^\#)}\right)\cdot\left(1-\prod_{l=1}^{k'}\left(1-\frac{c_l}{c(S^\#)}\right)\right)\right]\Gamma(S^\#)\\
\ge&\left[1-\left(1-\frac{c(S^\#)}{(k'+1)c(S^\#)}\right)\right]^{k'+1}\Gamma(S^\#) \ge \left(1-\frac{1}{e}\right)\Gamma(S^\#).
\end{aligned}$$

This completes the proof of Inequality (5).

Proof of Theorem 3.3. The proof follows closely the proof of Theorem 3 in [29]. First, the classic Chernoff concentration bound is presented as follows:

Lemma 3.5. *Suppose that X_i are independent random variables satisfying $X_i \in [0,1]$ for $1 \le i \le \omega$. Let $X = \sum_{i=1}^{\omega} X_i$ and $\mu = \mathbb{E}[X_i]$. Then, for any $\lambda > 0$*

$$Pr[X \le (1-\lambda)\cdot\omega\mu] \le \exp\left(-\frac{\lambda^2}{2}\cdot\omega\mu\right), \tag{10}$$

$$Pr[X \ge (1+\lambda)\cdot\omega\mu] \le \exp\left(-\frac{\lambda^2}{2+\lambda}\cdot\omega\mu\right). \tag{11}$$

Let $\Gamma_2(S^o) = \frac{\Lambda_{\mathcal{R}_2}(S^o)}{|\mathcal{R}_2|}\cdot|V|$ and $\Gamma_1(S^*) = \frac{\Lambda_{\mathcal{R}_1}(S^*)}{|\mathcal{R}_1|}\cdot|V|$. The variables ϵ_1 and ϵ_2 are defined in Procedure VerifySeedSet. Then we prove the following lemma.

Lemma 3.6. *With at least probability $1-\frac{5}{9n}$ it holds that*

$$\Gamma_2(S^o) \le (1+\epsilon_1)\cdot\Gamma(S^o), \tag{12}$$

$$\Gamma_1(S^*) \ge (1-\epsilon_2)\cdot\Gamma(S^*). \tag{13}$$

where $\epsilon_1, \epsilon_2 > 0$, for all iterations of Alg. 3.

Proof. Let ω_i be the size of $|\mathcal{R}_1|$ (recall that $|\mathcal{R}_1| = |\mathcal{R}_2|$), and let $\delta_i = \frac{\delta}{6i^2}$. Define λ_1 as the root to the following equation:

$$\exp\left(-\frac{\lambda^2}{2+\lambda}\cdot\frac{\Gamma(S^o)}{n}\cdot\omega_i\right) = \delta_i. \tag{14}$$

By Inequality (11), the following event $\mathcal{H}_1$ occurs with probability at least $1-\delta_i$:

$$\mathcal{H}_1 : \Gamma_2(S^o) < (1+\lambda_1)\cdot\Gamma(S^o). \tag{15}$$

Since $\Gamma(S^o)$ is unknown, and we cannot calculate the value of λ_1, we need to identify a value $\lambda' > \lambda_1$. When event $\mathcal{H}_1$ occurs, we can rewrite Equation. (14) as:

$$\begin{aligned}
\ln(1/\delta_i) &= \frac{\lambda_1^2}{2+\lambda_1}\cdot\frac{\omega_i}{n}\cdot\Gamma(S^o)\\
&> \frac{\lambda_1^2}{2+\lambda_1}\cdot\frac{\omega_i}{n}\cdot\frac{\Gamma_2(S^o)}{1+\lambda_1}.
\end{aligned} \tag{16}$$

Let λ' be the larger root of the equation:

$$(2+x)(1+x) = x^2\cdot\frac{\omega_i}{n\ln(1/\delta_i)}\cdot\Gamma_2(S^o).$$

We can infer that $\lambda' > \lambda_1$, because otherwise, λ_1 would not satisfy Inequality (16). According to the definition of ϵ_1 in the pseudo code, it follows that $\epsilon_1 = \lambda'$. Therefore, given that $\mathcal{H}_1$ occurs, we deterministically obtain $\Gamma_2(S^o) < (1+\epsilon_1)\cdot\Gamma(S^o)$ from $\Gamma_2(S^o) < (1+\lambda_1)\cdot\Gamma(S^o)$. This implies that Inequality (12) holds with probability at least $1-\delta_i$ for the i-th iteration.

Next, let λ_2 denote the root to the equation:

$$\exp\left(-\frac{\lambda^2}{2}\cdot\omega_i\cdot\frac{\Gamma(S^*)}{n}\right) = \delta_i. \tag{17}$$

Substituting λ_2 into the equation, we can rewrite it as:

$$\frac{\lambda_2^2}{2} = \ln(1/\delta_i)\cdot\frac{n}{\Gamma(S^*)\cdot\omega_i}. \tag{18}$$

Define the event $\mathcal{H}_2$ in the i-th iteration as:

$$\mathcal{H}_2 : \Gamma_1(S^*) \ge (1-\lambda_2)\cdot\Gamma(S^*).$$

By the Chernoff bound, we know that $\mathcal{H}_2$ occurs with probability at least $1-\delta_i$. Similarly, Since $\Gamma(S^*)$ is unknown, we need to find a value $\lambda'' > \lambda_2$. When both $\mathcal{H}_1$ and $\mathcal{H}_2$ hold, we have:

$$\begin{aligned}
\frac{\lambda_2^2}{2\cdot(1+\epsilon_1)} &= \frac{\ln(1/\delta_i)\cdot n}{(1+\epsilon_1)\cdot\Gamma(S^*)\cdot\omega_i}\\
&\le \frac{\ln(1/\delta_i)\cdot n}{(1+\epsilon_1)\cdot(\Gamma(S^*)-c(S^*))\cdot\omega_i}\\
&\le \frac{\ln(1/\delta_i)\cdot n}{(1+\epsilon_1)\cdot(\Gamma(S^o)-c(S^o))\cdot\omega_i}\\
&\le \frac{\ln(1/\delta_i)\cdot n}{(\Gamma_2(S^o)-(1+\epsilon_1)c(S^o))\cdot\omega_i},
\end{aligned}$$

where the second inequality follows from the optimality of S^*, and the third inequality follows from the occurrence of event $\mathcal{H}_1$.

Let λ'' be the larger root of:

$$\frac{x^2}{2\cdot(1+\epsilon_1)} = \frac{\ln(1/\delta_i)\cdot n}{(\Gamma_2(S^o)-(1+\epsilon_1)c(S^o))\cdot\omega_i}.$$

We can infer that $\lambda'' > \lambda_2$. From the definition of ϵ_2 in the pseudo code, we have $\epsilon_2 = \lambda''$. Therefore, given that $\mathcal{H}_1$ occurs, we obtain $\Gamma_1(S^*) \ge (1-\epsilon_2)\cdot\Gamma(S^*)$ from event $\mathcal{H}_2$. Recall that $\mathcal{H}_2$ occurs with probability at least $1-\delta_i$. Hence, Inequality (21) holds with probability at least $1-\delta_i$ given $\mathcal{H}_1$.

By the union bound, from the above discussions, we conclude that for all iterations, Inequalities (12) and (13) hold with the following probability:

$$1-\sum_{i=1}^{\infty}2\delta_i = 1-\sum_{i=1}^{\infty}\frac{\delta}{3i^2} = 1-\frac{\pi^2\cdot\delta}{18} \ge 1-\frac{5}{9}\cdot\delta, \tag{19}$$

where the second equality follows from the infinite series $\sum_{i=1}^{\infty}\frac{1}{i^2}=\frac{\pi^2}{6}$. This completes the proof. □

Now we give the proof of Theorem 3.3. For simplicity, we define $\alpha=(1-\frac{1}{e})/2$. We consider the following cases.

Case 1. Line 9 is triggered. In this case, either of the following events happens:

$$\mathcal{E}_1: t>1 \text{ and } (t-1)/t+\epsilon_1+\epsilon_2\le\epsilon, \quad (20)$$

$$\mathcal{E}_2: 0\le t\le 1 \text{ and } \epsilon_1+\epsilon_2\le\epsilon. \quad (21)$$

As $\Gamma_1(S)$ is also monotone and submodular, according to Theorem 3.1, it has

$$\begin{aligned}\Gamma_1(S^o)-c(S^o)&\ge\alpha\cdot\Gamma_1(S^*)-\frac{1}{2}\cdot c(S^*)\\&\ge(1-\epsilon_2)\cdot\alpha\cdot\Gamma(S^*)-\frac{1}{2}\cdot c(S^*),\end{aligned}\quad (22)$$

where the second inequality comes from Inequality (13). Then,

$$\begin{aligned}\Gamma_1(S^o)-c(S^o)&=t\left(\Gamma_2(S^o)-c(S^o)\right)\\&\le t\left(\Gamma(S^o)-c(S^o)\right)+t\epsilon_1\Gamma(S^o).\end{aligned}$$

where the inequality is due to Inequality (12). Re-organize the above inequality, we obtain

$$\Gamma(S^o)-c(S^o)\ge\frac{1}{t}\cdot\left(\Gamma_1(S^o)-c(S^o)\right)-\epsilon_1\Gamma(S^o) \quad (23)$$

$$\ge\frac{1}{t}\cdot\left((1-\epsilon_2)\cdot\alpha\cdot\Gamma(S^*)-\frac{1}{2}\cdot c(S^*)\right)-\epsilon_1\Gamma(S^o)$$

$$\ge\frac{1}{t}\cdot\left((1-\epsilon_2)\cdot\alpha\cdot\Gamma(S^*)-\frac{1}{2}\cdot c(S^*)\right)-\epsilon_1\alpha\cdot\Gamma(S^*). \quad (24)$$

Next, we will consider two cases: $t>1$ and $t\le 1$. If $t>1$, then by Eq. (24), we have

$$\Gamma(S^o)-c(S^o)\ge\frac{1}{t}\cdot\left((1-\epsilon_2)\cdot\alpha\cdot\Gamma(S^*)-\frac{1}{2}\cdot c(S^*)\right)-\epsilon_1\alpha\cdot\Gamma(S^*)$$

$$\ge\alpha\cdot\Gamma(S^*)-\frac{1}{2}\cdot c(S^*)-\frac{t-1}{t}\cdot\alpha\cdot\Gamma(S^*)-(\epsilon_1+\epsilon_2)\cdot\alpha\cdot\Gamma(S^*)$$

$$\ge(1-\epsilon)\cdot\alpha\cdot\Gamma(S^*)-\frac{1}{2}\cdot c(S^*),$$

where the last inequality is due to Inequality (20).

On the other side, if $t\le 1$, Inequality (24) becomes

$$\begin{aligned}&\Gamma(S^o)-c(S^o)\\&\ge\left((1-\epsilon_2)\cdot\alpha\cdot\Gamma(S^*)-\frac{1}{2}\cdot c(S^*)\right)-\epsilon_1\alpha\cdot\Gamma(S^*)\\&\ge(1-\epsilon)\cdot\alpha\cdot\Gamma(S^*)-\frac{1}{2}\cdot c(S^*),\end{aligned}$$

where the last inequality comes from Inequality (21).

Therefore, for both case of $t>1$ and $t\le 1$, we have $\Gamma(S^o)-c(S^o)\ge\frac{1}{2}\left((1-1/e)(1-\epsilon)\cdot\Gamma(S^*)-c(S^*)\right)$.

Case 2. As Line 9 is not triggered, Alg. 3 is terminated by Line 10. In this case, the size of $\mathcal{R}_1$ is

$$|\mathcal{R}_1|\ge\omega_{\max}=\frac{(8+2\alpha\epsilon)\cdot(1+\epsilon_1)\cdot n\cdot(\ln(6/\delta)+n\ln 2)}{\epsilon^2\cdot\alpha^2\cdot\max\{1,\Gamma_2(S^o)-(1+\epsilon_1)c(S^o)\}}.$$

Conditioned on that Inequality (12) holds for all iterations, it has

$$\begin{aligned}&\max\left\{1,\Gamma_2(S^o)-(1+\epsilon_1)c(S^o)\right\}\\&\le\max\left\{1,(1+\epsilon_1)\left(\Gamma(S^o)-c(S^o)\right)\right\}\\&\le\max\left\{1,(1+\epsilon_1)\Gamma(S^*)\right\}\\&=(1+\epsilon_1)\Gamma(S^*),\end{aligned}$$

where the last equality is due to $\Gamma(S^*)\ge 1$ for any non-empty S^* and $\epsilon_1>0$. Therefore, we have

$$\omega_{\max}\ge\frac{(8+2\alpha\epsilon)\cdot n\cdot(\ln(6/\delta)+n\ln 2)}{\epsilon^2\cdot\alpha^2\cdot\Gamma(S^*)}.$$

Then, by Chernoff concentration bound, for any $S\subset V$, when $|\mathcal{R}_1|\ge\omega_{\max}$, the probability

$$\Pr\left[|\Gamma_1(S)-\Gamma(S^*)|\le\frac{\epsilon\cdot\alpha}{2}\Gamma(S^*)\right]\le\frac{\delta}{3\cdot 2^n}.$$

As there exist 2^n possible S, by union bound, the total failure probability will be bounded by $\frac{\delta}{3}$. Thus, with probability at least $1-\frac{\delta}{3}$, we have

$$\Gamma_1(S^*)\ge(1-\frac{\alpha\cdot\epsilon}{2})\Gamma(S^*)\ge(1-\frac{\epsilon}{2})\Gamma(S^*),$$

$$\Gamma_1(S^o)\le\Gamma(S^o)+\frac{\epsilon\cdot\alpha}{2}\Gamma(S^*).$$

From the above equations, we further derive that

$$\begin{aligned}&\Gamma(S^o)-c(S^o)\\&\ge\Gamma_1(S^o)-c(S^o)-\frac{\epsilon\cdot\alpha}{2}\Gamma(S^*)\\&\ge\alpha\cdot\Gamma_1(S^*)-\frac{1}{2}c(S^*)-\frac{\epsilon\cdot\alpha}{2}\Gamma(S^*)\\&\ge\alpha\cdot(1-\frac{\epsilon}{2})\Gamma(S^*)-\frac{1}{2}c(S^*)-\frac{\epsilon\cdot\alpha}{2}\Gamma(S^*)\\&\ge\alpha\cdot(1-\epsilon)\Gamma(S^*)-\frac{1}{2}c(S^*)\\&\ge\frac{1}{2}(1-\epsilon)(1-\frac{1}{e})\Gamma(S^*)-\frac{1}{2}c(S^*).\end{aligned}$$

Next, we consider the total failure probability of Alg. 3. According to Lemma 3.6, the probability that Alg. 3 is terminated by Line 9 while returning an invalid seed set is $\frac{5\delta}{9}$. When Alg. 3 is terminated by Line 10, it returns an invalid seed set with probability $\frac{\delta}{3}$. Therefore, Alg. 3 returns a good seed set with probability at least $1-\frac{5\delta}{9}-\frac{\delta}{3}\ge 1-\delta$.

Proof of Lemma 3.4 The time complexity of RAP consists of two parts: the first is the cost of generating RR-sets, and the second is the cost of seed selection and verification. For generating RR-sets, from Line 10, we know that the maximum number of RR-sets is $O\left(\frac{n\cdot\max\{\ln(1/\delta),n\}}{\epsilon^2}\right)$. Assuming that the expected running time for generating a random RR-set is τ, the cost of generating the RR-sets is $O\left(\frac{n\cdot\tau\cdot\max\{\ln(1/\delta),n\}}{\epsilon^2}\right)$.

On the other hand, the process of verifying the seed set is bounded by the cost of the RR-sets. For seed selection, we use the lazy update strategy, where the update cost is also bounded by the RR-sets. In each iteration of the greedy selection process, the total cost is $O\left(n\cdot\frac{B}{c_{\min}}\right)$, where $c_{\min}$ denotes the minimum cost of a node in the graph. This cost is also bounded by $O\left(n\cdot\max\{\ln(1/\delta),n\}\right)$.

Thus, the overall time complexity of RAP is $O\left(\frac{n\cdot\tau\cdot\max\{\ln(1/\delta),n\}}{\epsilon^2}\right)$.

4 New RR-set Generation

Recall that in RAP, we maintain two sets of random RR-sets, i.e., $\mathcal{R}_1$ and $\mathcal{R}_2$. We continuously double their size until a qualified seed solution is found. Therefore, the phase of generating random RR-sets plays a critical role in the efficiency of RAP. In this section, we introduce a new RR-set generation method, termed RISC[3], which achieves a significant speedup compared to existing methods.

4.1 SUBSIM process breakdown

Before presenting our new RR-set generation method, we first decompose the SUBSIM process into several stages. This breakdown allows us to gain detailed insights into when memory accesses occur within SUBSIM. For simplicity, we omit the initialization step, which involves sampling a target node and enqueuing it into the queue. The detailed breakdown is as follows:

- **Stage 1** (denoted as T_1): Retrieve the node ID of the current frontier node from queue Q, and then load the meta-information of its adjacency list, such as the starting address of the list and the number of in-neighbors.
- **Stage 2** (T_2): Use geometric distribution sampling to select a subset of in-neighbors according to edge probability. This stage consists of three substages, listed as follows:
  - **Substage 2.1** (T_{21}): Calculate the successfully sampled positions in the adjacency list of u (see Lines 9–12 of Alg. 1). Then, load the IDs, denoted as W, of the sampled nodes from the adjacency list.
  - **Substage 2.2** (T_{22}): Load the activation statuses of the sampled nodes in W in Substage T_{21}.
  - **Substage 2.3** (T_{23}): For each $w \in W$, if w is not activated, update the RR-set R and enqueue w in queue Q.
- **Stage 3** (T_3): If Q is empty, the RR-set R is deemed complete; If Q is not empty, go to Stage T_1.

In previous work, when a set $\mathcal{R}$ of random RR-sets are required, these random RR-sets are generated sequentially (assuming a single working thread). The generation of a random RR-set involves stage switching in the following order: $T_1 \longrightarrow T_{21} \longrightarrow T_{22} \longrightarrow T_{23} \longrightarrow T_3 \cdots$. Note that at the end of stages T_1, T_{21}, and T_{22}, memory access operations occur. For instance, in Stage T_1, the meta-information of the adjacency list for frontier u should be loaded. Since this is executed sequentially, the CPU is forced to stall until the data arrives, resulting in high memory access latency.

4.2 RISC

Instead of generating a set of random RR-sets one by one, our new RR-set generation method RISC works in a batch manner. Let b denote the batch size of RISC, and R_i denote the i-th RR-set in the batch. As shown in Figure 2, we present the workflow diagram of our RISC for illustration, where the batch size b is 3. Note that it is single-threaded execution. When this batched execution completes, we obtain the set $\{R_1, R_2, R_3\}$.

In particular, when the CPU is processing Stage T_1 of R_1, we need to load the meta-information of the current frontier u in preparation for Stage 2. Instead of stalling the CPU until the data arrives, we issue an asynchronous data prefetching request (denoted as a dashed arrow in Figure 2) to bring the target data into the cache system, and immediately proceed to process Stage T_1 of R_2. The data prefetch request can be implemented by inserting prefetch instructions in the source code (e.g., API *_mm_prefetch* in C++). Similarly, after completing Stage T_1 of R_2, we move on to Stage T_1 of R_3. Once Stage T_1 of R_3 is completed, we have finished processing Stage T_1 for all RR-sets in the batch. We then proceed to Stage T_{21} of R_1. Since we have prefetched the meta-information of the frontier into the cache, this allows us to quickly retrieve the data and continue processing Stage T_{21} of R_1. Again, at the end of Stage T_{21}, we issue another prefetching request and move on to process Stage T_{21} of R_2. By this batched manner, we hide the memory latency during the processing of other RR-sets, effectively speeding up the generation of random RR-sets.

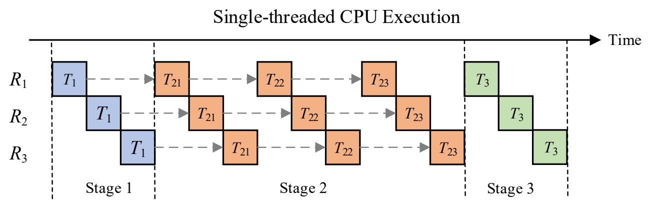


Figure 2: A workflow diagram of RISC with batch size $b = 3$. The *dashed arrow* represents cache prefetching.

The pseudo code of RISC is presented in Alg. 4. The algorithm begins by initializing each RR-set R_k and its corresponding queue Q_k, where $k = 1, \ldots, b$. All RR-sets are initially marked as incomplete by setting the flags $idle[k] = false$. The execution then enters a while loop. Within each iteration, we process the RR-sets through the stages T_1, T_{21}, T_{22}, T_{23}, and T_3, as detailed in Section 4.1. An RR-set R_k is marked as completed if its corresponding queue Q_k is empty, and it does not participate in subsequent iterations of the while loop. This loop continues until all RR-sets have been processed. Note that the activation status of any node $v \in V$ when generating R_i should be independent of its status for any R_j, where $i \neq j$. Hence, we maintain independent activation statuses for all nodes for each individual R_k, with $k \in [1, \ldots, b]$.

Unbiased Estimation Analysis. Within the same batch, the generation of R_i is independent of that of any other R_j, where $i \neq j$. Consequently, the RR-sets within the same batch are independent. Furthermore, RR-sets generated in different batches are assuredly independent. Therefore, when using the algorithm RISC to generate a set $\mathcal{R}$ of random RR-sets, the estimator $n \cdot \frac{\Lambda_{\mathcal{R}}(S)}{|\mathcal{R}|}$ remains an unbiased estimator of the expected value $\Gamma(S)$, ensuring its applicability.

Choice of batch size b. The parameter of batch size b determines how many RR-sets are executed simultaneously. Due to hardware limitations, such as the size of the L1 cache, the value of b is constrained. To identify the most effective setting, we tune b by varying it over the set $\{2, 4, 8, \ldots\}$. We then select the optimal value that yields the largest efficiency improvement. In the experimental section (i.e., Section 6), we present the results and demonstrate that $b = 16$ is a suitable choice for our equipment.

[3]Batched Reverse Influence Sampling with Cache Prefetching

Algorithm 4: RISC

```
Input: Input graph $G = (V, E)$, batch size $b$
Output: Set of RR-sets, $\mathcal{R}_b = \{R_1, R_2, \ldots, R_b\}$
Let $idle[k]$ represent whether $R_k$ is completed;
foreach $k \in [1, b]$ do
    Initialize $R_k$ and queue $Q_k$;
    $idle[k] \leftarrow false$                      // $R_k$ is not completed
end
while there exists any working $R_k \in \mathcal{R}_b$ do
    $K \leftarrow \{k \in [1, b] \mid idle[k] == false\}$;
    foreach $k \in K$ do                            // Stage $T_1$
        $u_k \leftarrow$ pop the top element of $Q_k$;
        Prefetch meta-information of $u_k$'s adjacency list;
    end
    foreach $k \in K$ do                            // Stage $T_{21}$
        $\mathcal{I}_k \leftarrow$ calculate the sampled positions for frontier $u_k$;
        Prefetch node IDs for set $\mathcal{I}_k$ from $u_k$'s adjacency list;
    end
    foreach $k \in K$ do                            // Stage $T_{22}$
        Let $W_k$ be the set of node IDs according to the
          position set $\mathcal{I}_k$;
        Prefetch the activation status for each $w \in W_k$;
    end
    foreach $k \in K$ do                            // Stage $T_{23}$
        foreach $w \in W_k$ and $w$ is not activated do
            Add $w$ into $R_k$, and enqueue $w$ into $Q_k$;
            Mark $w$ as activated;
        end
    end
    foreach $k \in K$ do                            // Stage $T_3$
        if $Q_k$ is empty then $idle[k] \leftarrow true$;
    end
end
return $\mathcal{R}_b$
```

Space Usage. Compared to SUBSIM, RISC needs to maintain b boolean arrays to track the activation status and b queues. Each array spans a length of n, thereby requiring an additional $\frac{b}{8} \times n$ bytes for the boolean arrays, as each boolean value occupies one bit. In contrast, the storage space for the input graph is calculated as $4n + 8m$ bytes, assuming that nodes are represented by integers and edge probabilities by floating-point types. Therefore, when b is a small value (e.g., $b = 16$) and $m > n$, the additional space required for the b boolean arrays is negligible compared to the storage of the graph. Regarding the b queues, their space is predominantly governed by that of the set $\mathcal{R}$, where typically $|\mathcal{R}| \gg b$. Consequently, RISC results in only a slight increase in memory usage.

Extension to general IC model. For the general IC model, the probabilities of incoming edges to a node are skewed. According to the extended work of SUBSIM [24], to ensure the correctness of the sampling probabilities, a rejection operation is performed immediately after each geometric distribution sampling. By similarly performing the rejection operations, our RISC method can be adapted to support the general IC model. In Section 6, we conduct experiments to explore this extension further.

Table 1: Datasets. ($M = 10^6, B = 10^9$)

Abbr.	Name	n	m	Type
PO	*Pokec*	1.6M	30.6M	directed
LJ	*LiveJournal*	4.8M	69.0M	directed
OR	*Orkut*	3.1M	117.2M	undirected
TW	*Twitter*	65.6M	1.8B	directed

5 Additional Related Works

The Influence Maximization (IM) problem has been extensively explored over the past decades in the literature [7, 8, 12, 14–16, 20, 21, 31]. Kempe et al. [31] present the first seminal work on the IM problem, proving that identifying k users to maximize influence is NP-hard. They further propose a greedy algorithm that provides a $(1 - \frac{1}{e})$-approximation. Then, a large number of subsequent research focus on improving the efficiency of solving the IM problem [14, 15, 23, 44, 47, 48], while still providing approximation guarantee. In addition, a plethora of works focus on more practical scenarios beyond the classic IM, such as topic-aware IM [32, 39], time-aware IM [34], and budgeted IM [8, 22, 39]. The profit maximization (PM) problem is proposed, aiming to maximize net profit, which is a more practical consideration in applications. Besides the works [10, 25, 29, 30, 36, 46] mentioned in Section 2.2, there are other studies on the PM problem under various settings. For instance, adaptive PM [26] selects seed users across multiple rounds, and streaming PM [17] addresses scenarios where elements may arrive sequentially in any order. These are orthogonal to our work.

In the literature, several approaches have been proposed to enhance the efficiency of RR-set-based solutions. In addition to SUBSIM [23], which accelerates RR-set generation, another line of research focuses on reducing the number of RR-sets required while maintaining the same approximation guarantees, as demonstrated in [8, 44, 47, 48]. These methods typically rely on deriving tighter concentration bounds. Additionally, in the adaptive IM setting, Guo et al. [22] have designed an approach that incrementally updates and reuses RR-sets, providing accurate estimations even when the graph is partially influenced.

On the other hand, cache prefetching is an effective tool for reducing memory latency when retrieving data from outside of computational cores. This technique has been extensively explored in various fields, including hash joins [11], B$^+$ Trees [13], Masstree [37], random walk engines [43], and query processing on new hardware, such as GPUs [18] and NVMs [41]. Our work is the first to apply cache prefetching to RR-set generation.

6 Experiments

In this section, we experimentally evaluate the performance of our algorithms (i.e., GEAR, RAP, and RISC) against alternatives. All codes are implemented in C++ language [4], and all experiments are run on a Linux server with Intel Xeon Gold 6248 CPU, which is clocked at 2.5 GHz. The size of L1, L2, and L3 cache is 32 KB, 1 MB, and 27.5 MB, respectively.

Datasets. Four large-scale real datasets of social networks are used to evaluate algorithm performance: Pokec, LiveJournal, Orkut, and Twitter. They are commonly used in previous related works [23, 46], and are publicly available in [1, 2]. We have summarized their statistic information in Table 1. The largest dataset Twitter consists of 65.6 million nodes and 1.8 billion edges, which allows us to test the scalability of our solutions.

Cost Model. In our experiments, we adopt the degree-based cost model, following prior works [22, 29]. Specifically, for a node $v \in V$, the cost to select it as a seed is directly proportional to its number of in-neighbors plus one, expressed as $c_v = c \cdot (d_{\text{in}}(v) + 1)$, where $d_{\text{in}}(v)$ represents the in-degree of node v, and c is a constant number adjusting the cost model [29]. This model represents that influencers with a larger number of followers typically demand higher prices for promotional activities [28, 42]. The addition of the constant 1 ensures that the cost is set non-zero even for those nodes without in-neighbors. In the experiments, we set $c = 0.005$.

Algorithms. We evaluate our GEAR against the alternatives Simple Greedy [36], Distorted Greedy [25], and Sample Greedy [6]. For short, we denote them as SIG, DIG, and SAG, respectively. The alternatives Simple Greedy and Distorted Greedy are adapted to handle the budget constraint, by rejecting the locally optimal nodes if adding the nodes exceeds budget B. Recall that our GEAR algorithm mainly adopts the same cost-effective greedy strategy for node selection as ROI Greedy [29]. To address the budgeted constraint, GEAR further integrates node rejection and the singleton node set. Thus, our GEAR can be considered a logical extension of ROI Greedy to the budgeted scenario. More importantly, we have rigorously proven that such an extension provides a strong approximation guarantee for the BPM problem, which constitutes our main contribution. Therefore, we will not pursue other adaptations of ROI Greedy for the budget setting. Furthermore, since the algorithm SAG is randomized and provides an expected approximation guarantee, we execute it five times and report the average result for each experiment. In addition, for RR-set generation, we compare our RISC solution with the baseline method SUBSIM, using its GitHub implementation [3].

Parameter setting. We adopt the IC model for the BPM problem. Following prior works [23, 26, 29, 44], the diffusion probability of each edge (v, u) is set to be $\frac{1}{d_{\text{in}}(u)}$. The budget B varies across the set $\{100, 200, 300, 500, 1000\}$. For algorithm RAP, we set the error threshold parameter $\epsilon = 0.1$ and failure probability $\delta = 1/n$. The batch size b of RISC is set as $b = 16$. To calculate the expected influence spread, we conduct 10000 Monte Carlo simulations [21] and take the average.

6.1 Effectiveness of GEAR

We first conduct experiments to assess the profit performance of our GEAR compared to its competitors with varying budget B. As the exact expected influence spread $\Gamma(S)$ is unknown, we generate a set of $2^{10} \times 10^4$ random RR-sets (using SUBSIM) to estimate the expected influence spread. We use the profits obtained by GEAR as a benchmark and then report the profit ratios of baseline algorithms relative to GEAR. If a baseline algorithm returns a seed set with a profit ratio smaller than 1, it indicates that GEAR outperforms it in terms of profit performance, and vice versa.

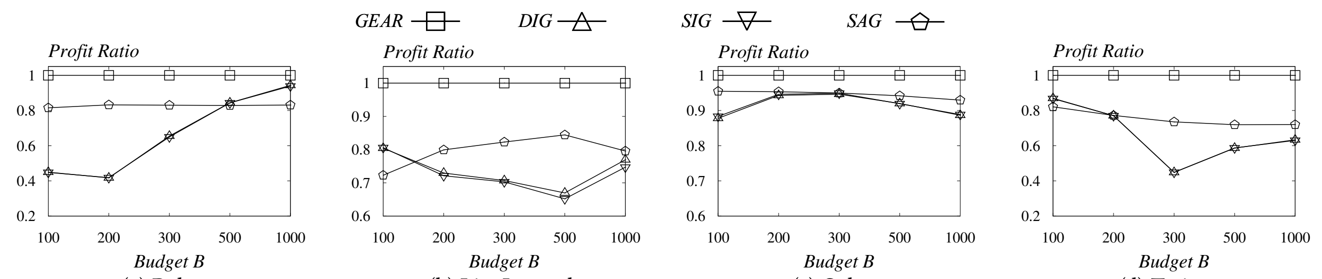
The experimental results of the profit performance are shown in Figure 3. Firstly, note that none of the data points across all datasets exceed a value of 1, demonstrating that GEAR consistently achieves the highest profit. The baselines SIG, DIG, and SAG typically generate seed sets that yield significantly lower profits than those achieved by GEAR. Specifically, SAG, which is designed for non-negative, non-monotone submodular problems under knapsack constraints, consistently produces a stable, albeit lower, profit performance compared to GEAR. Across all four datasets, SAG generates seed sets achieving at least 70% of the profit of GEAR. As both algorithms employ a similar influence-to-cost seed selection strategy, the reduced performance of SAG can be attributed to its strategy of selecting seeds from a sampled subset, which impacts its profit potential. For the other two baselines, SIG and DIG, their performance can be as low as 40% of that of GEAR in certain budget scenarios (e.g., $B = 200$ on dataset Pokec, and $B = 300$ on dataset Twitter). This lower performance is due to these two algorithms favoring nodes with higher influence without adequately considering the associated costs. Consequently, they may deplete their budget after selecting only a few high-cost nodes, resulting in a significantly low total profit. Neither SIG nor DIG provides a theoretical approximation guarantee for the BPM problem.

Next, we examine the impact of the number of random RR-sets on the profit performance of each algorithm, setting the budget $B = 200$. The number of random RR-sets varies from $2^0 \times 10^4$ to $2^{10} \times 10^4$. We benchmark the profit value of our GEAR obtained with $2^{10} \times 10^4$ RR-sets on each dataset and report the profit ratio. The experimental results are displayed in Figure 4. It is observed that as the number of RR-sets approaches $2^{10} \times 10^4$, the profit obtained by each algorithm converges to stable values. This observation justifies the setting of $2^{10} \times 10^4$ random RR-sets to evaluate the profit performance for different budgets in our first set of experiments. Furthermore, our solution GEAR consistently outperforms the three baseline algorithms when the number of RR-sets exceeds $2^4 \times 10^4$ across all datasets. However, in the case of a smaller number of RR-sets (e.g., $2^0 \times 10^4$ on dataset LiveJournal), GEAR may perform worse than SIG and DIG. This is because the influence-to-cost strategy is sensitive to the accuracy of the influence estimation. Therefore, an estimation with significant deviation from true values may result in a seed set that generates lower profits.

6.2 Effectiveness of RAP

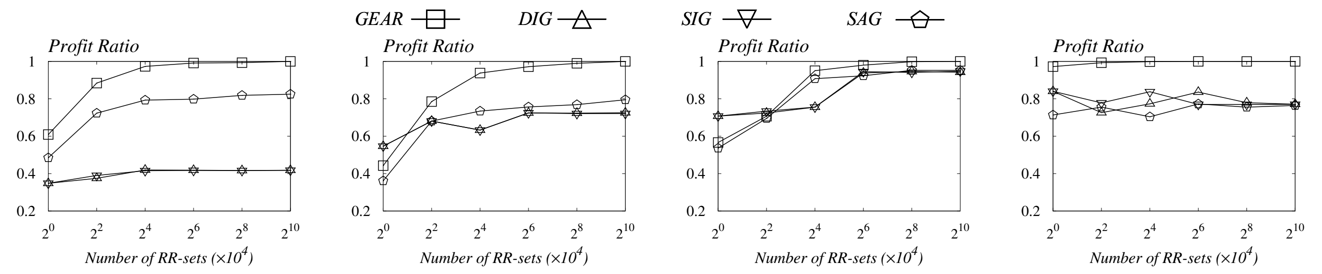
In the experiments described above, we observe that the profit performance of GEAR may deteriorate if the number of RR-sets is insufficient to provide an accurate influence estimation. The requisite number of RR-sets can vary across different datasets. Fortunately, our algorithm, RAP, presented in Section 3.2, can address this issue. It outputs a seed solution with a strong approximation guarantee without the need to manually specify the required number of RR-sets for accurate influence estimation.

In this section we examine the performance of RAP (instantiated with SUBSIM for RR-set generation) in terms of profit and running time. When presenting the profit metric, we benchmark the profit values of GEAR obtained with $2^{10} \times 10^4$ RR-sets, and



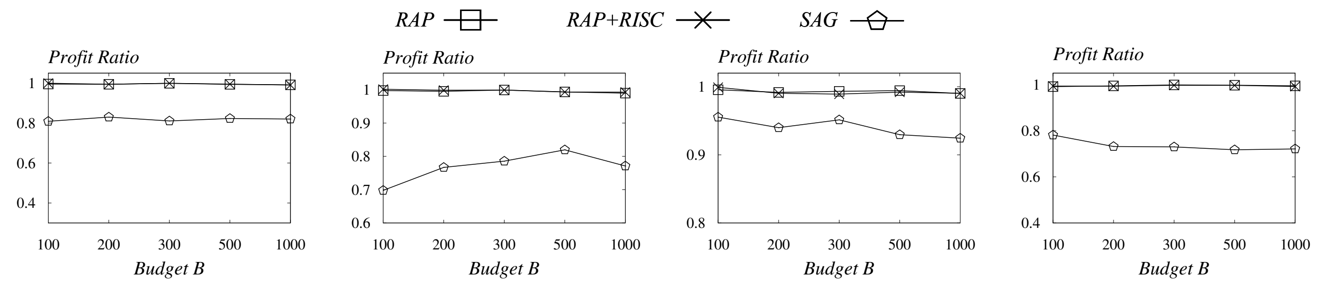
(a) Pokec (b) LiveJournal (c) Orkut (d) Twitter

Figure 3: Varying budget B: Profit performance of different algorithms.



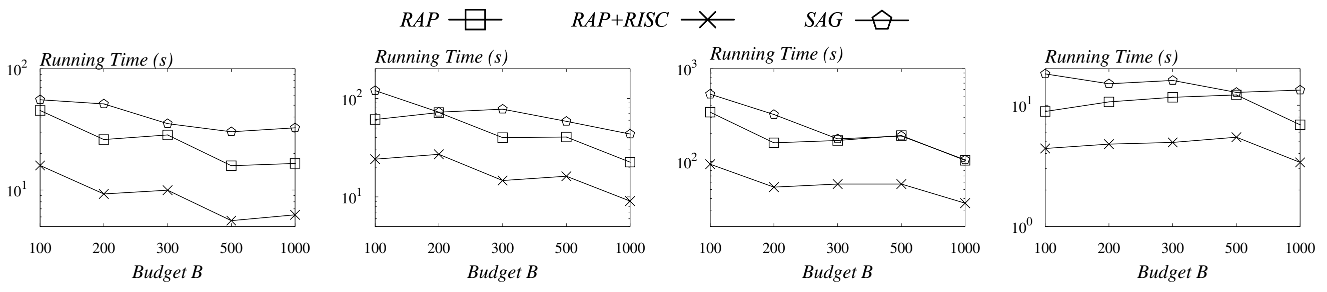
(a) Pokec (b) LiveJournal (c) Orkut (d) Twitter

Figure 4: Varying the number of RR-sets: Profit performance of different algorithms.



(a) Pokec (b) LiveJournal (c) Orkut (d) Twitter

Figure 5: Varying budget B: Profit performance of RAP with $\epsilon = 0.1$.

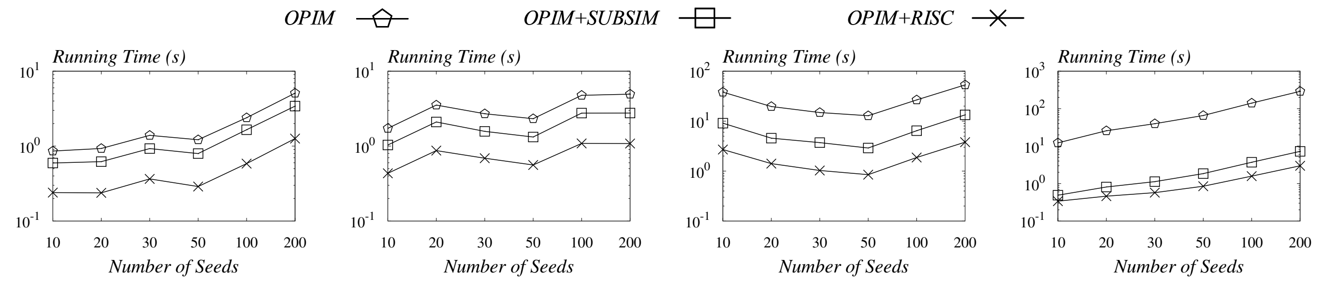


(a) Pokec (b) LiveJournal (c) Orkut (d) Twitter

Figure 6: Varying budget B: Running Time of RAP with $\epsilon = 0.1$.

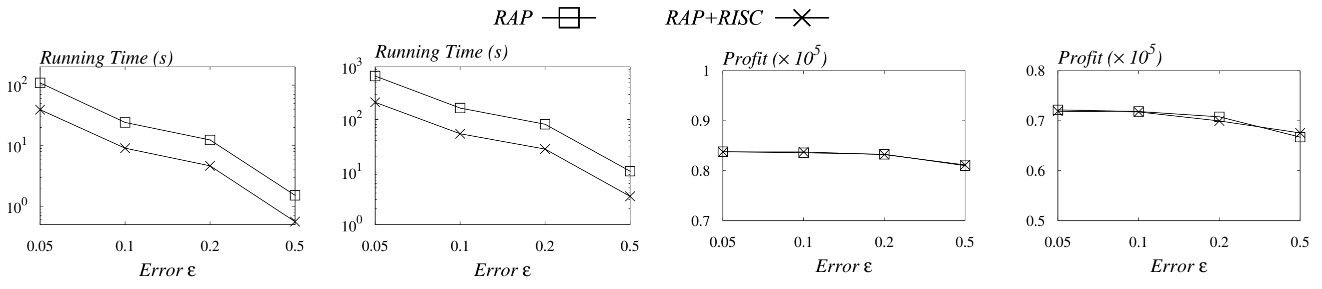
report the profit ratio again. For comparison, we include a baseline that adopts Sample Greedy for seed selection (see Line 3 in Procedure SeedSelection), which is still denoted as SAG. On the other hand, as shown in Figure 3, Simple Greedy and Distorted Greedy provide a significantly worse outcome in some budget cases, and thus are not equipped into our RAP framework. In addition, we also equip RAP with our new RR-set method, denoted as RAP+RISC. The experimental results are shown in Figures 5–6.

In terms of profit performance, we observe from Figure 5 that the profit curves of RAP and RAP+RISC visually overlap. This overlap is expected, as the RR-sets generated by either SUBSIM or our RISC method both serve as an unbiased estimator for the influence spread.



(a) Pokec (b) LiveJournal (c) Orkut (d) Twitter

Figure 7: Running time of OPIM instantiated with different RR-set generation methods.



(a) Pokec (b) Orkut (c) Pokec (d) Orkut

Figure 8: Running time and profit performance of RAP with ϵ varying.

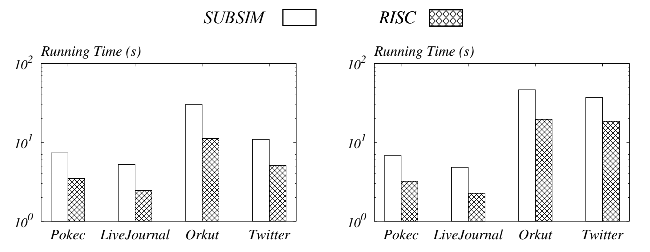
Conversely, SAG consistently shows lower profit performance than our solutions, attributed to its inferior seed selection.

Regarding running time, our proposed algorithm RAP demonstrates high efficiency and scalability. It can generate a seed set with the specified approximation guarantee within one minute on the billion-scale Twitter dataset. Additionally, the combined method RAP+RISC is the fastest among the three algorithms. For instance, RISC achieves a speedup of up to 3.2 times on the Orkut dataset compared to RAP where RR-sets are generated by SUBSIM. The efficiency improvement stems from our RISC method, which reduces memory stalls by concealing memory access latency during the batch execution of RR-sets. This demonstrates the effectiveness of our new RR-set generation method.

Furthermore, we examine the impact of the error parameter ϵ by varying ϵ from 0.5 to 0.05, only on the Pokec and Orkut datasets due to space constraints. The experimental results, shown in Figure 8, indicate that as ϵ increases, the running time for RAP and RAP+RISC decreases. This reduction occurs because a larger ϵ allows for a looser approximation guarantee, which leads to earlier terminations. However, this trade-off results in a slight degradation in the profit performance of the returned solution.

6.3 Effectiveness of RISC

To further explore the application of our RR-set generation method RISC, we conduct experiments on the classic IM problem [23, 44], which aims at identifying a set of k users with the largest expected influence. Tang et al. [44] propose the solution OPIM, which addresses the IM problem with a $(1-1/e-\epsilon)$ approximation guarantee. Subsequently, Guo et al. [23] enhance its efficiency with the improved RR-set generation method, SUBSIM. In our experiments,

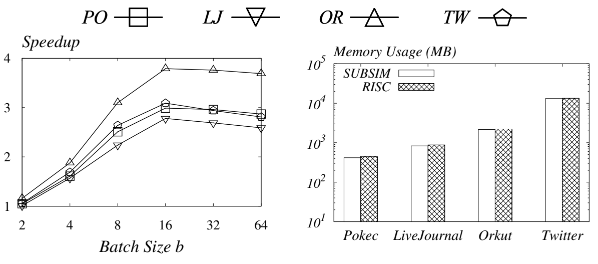


(a) Exponential (b) Weibull

Figure 9: Running time under IC model with a skewed probability distribution.

we implement OPIM with RISC and compare it against the aforementioned methods. We denote these implementations as OPIM, OPIM+SUBSIM, and OPIM+RISC, respectively. We exclude the solution IMM [47] as it is reported to be outperformed by OPIM in [23]. The size of seed set k varies from 10 to 200. The experimental results are illustrated in Figure. 7, showing that OPIM+RISC has the best performance among these three algorithms, achieving a speedup of up to 3.5x on the Orkut dataset. This indicates that our method can be used to accelerate existing RR-set-based solutions, greatly expanding its application scope.

Furthermore, we extend RISC to handle the general IC model, by including a rejection step to determine whether a node is actually sampled, addressing the skewed probability distribution. For modeling the edge probabilities in the general IC model, we adopt the configurations in [23]. Specifically, we draw the probabilities for each edge from the exponential distribution and Weibull distribution, respectively. These probabilities are then normalized such



(a) (b)

Figure 10: (a) Speedup of our RISC against SUBSIM with batch size b varying; (b) Memory Usage of RISC.

that the sum of the probabilities of all incoming edges from a node equals 1. Then, we generate $2^{10} \times 10^3$ RR-sets using either SUBSIM or RISC and report their running time. It is observed that in either exponential distribution or Weibull distribution, our RISC keeps outperforming SUBSIM on all datasets with a 2 ∼ 3x speedup.

Then, we examine the impact of the batch size on the efficiency of RISC. We measure the speedup of RISC against SUBSIM as the batch size b varies from 2 to 64. As shown in Figure 10(a), the optimal efficiency gains across four datasets are achieved when $b = 16$. Consequently, we set $b = 16$ for our experimental evaluations. Additionally, we explore the memory consumption of RISC, as shown in Figure 10(b), when we generate $2^{10} \times 10^3$ random RR-sets using either SUBSIM or RISC. We observe that the memory usages of SUBSIM and RISC are nearly identical, consistent with the space analysis presented in Section 4.

7 Conclusion

In this paper, we formally define the BPM problem by introducing a budget constraint to reflect the limitations of marketing costs in real-world scenarios. Assuming access to a value oracle for the expected influence spread, we propose GEAR, the first algorithm with a provable approximation guarantee for the BPM problem. We then integrate GEAR with the RR-set technique to develop RAP, a practical and efficient solution for BPM. Additionally, we introduce a new RR-set generation method that significantly accelerates RAP. Extensive experiments on real large-scale datasets demonstrate the effectiveness and efficiency of our algorithms.

References

[1] 2013. KONECT. http://konect.cc/networks/.

[2] 2014. SNAP Datasets. http://snap.stanford.edu/data.

[3] 2020. SUBSIM. https://github.com/qtguo/subsim.git.

[4] 2024. Technical Report and Codes. https://anonymous.4open.science/r/bpm-0201.

[5] Georgios Amanatidis, Federico Fusco, Philip Lazos, Stefano Leonardi, and Rebecca Reiffenhäuser. 2020. Fast Adaptive Non-Monotone Submodular Maximization Subject to a Knapsack Constraint. In *NeurIPS 2020*.

[6] Georgios Amanatidis, Federico Fusco, Philip Lazos, Stefano Leonardi, and Rebecca Reiffenhäuser. 2022. Fast Adaptive Non-Monotone Submodular Maximization Subject to a Knapsack Constraint. *J. Artif. Intell. Res.* 74 (2022), 661–690.

[7] Prithu Banerjee, Wei Chen, and Laks V. S. Lakshmanan. 2023. Mitigating Filter Bubbles Under a Competitive Diffusion Model. *Proc. ACM Manag. Data* 1, 2 (2023), 175:1–175:26.

[8] Song Bian, Qintian Guo, Sibo Wang, and Jeffrey Xu Yu. 2020. Efficient Algorithms for Budgeted Influence Maximization on Massive Social Networks. *Proc. VLDB Endow.* 13, 9 (2020), 1498–1510.

[9] Christian Borgs, Michael Brautbar, Jennifer T. Chayes, and Brendan Lucier. 2014. Maximizing Social Influence in Nearly Optimal Time. In *SODA*. 946–957.

[10] Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. 2012. A Tight Linear Time (1/2)-Approximation for Unconstrained Submodular Maximization. In *FOCS*. 649–658.

[11] Shimin Chen, Anastassia Ailamaki, Phillip B. Gibbons, and Todd C. Mowry. 2007. Improving hash join performance through prefetching. *ACM Trans. Database Syst.* 32, 3 (2007), 17.

[12] Shuo Chen, Ju Fan, Guoliang Li, Jianhua Feng, Kian-Lee Tan, and Jinhui Tang. 2015. Online Topic-Aware Influence Maximization. *Proc. VLDB Endow.* 8, 6 (2015), 666–677.

[13] Shimin Chen, Phillip B. Gibbons, and Todd C. Mowry. 2001. Improving Index Performance through Prefetching. In *SIGMOD*. 235–246.

[14] Wei Chen, Chi Wang, and Yajun Wang. 2010. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *SIGKDD*. 1029–1038.

[15] Wei Chen, Yajun Wang, and Siyu Yang. 2009. Efficient influence maximization in social networks. In *SIGKDD*. ACM, 199–208.

[16] Wei Chen, Yifei Yuan, and Li Zhang. 2010. Scalable Influence Maximization in Social Networks under the Linear Threshold Model. In *ICDM*. 88–97.

[17] Shuang Cui, Kai Han, Jing Tang, and He Huang. 2023. Constrained Subset Selection from Data Streams for Profit Maximization. In *WWW*. ACM, 1822–1831.

[18] Yangshen Deng, Shiwen Chen, Zhaoyang Hong, and Bo Tang. 2024. How Does Software Prefetching Work on GPU Query Processing?. In *DaMoN*. 5:1–5:9.

[19] Flaminjoy. 2022. Influencer Marketing Budgets. https://flaminjoy.com/blog/influencer-marketing-budget/.

[20] Amit Goyal, Francesco Bonchi, and Laks V. S. Lakshmanan. 2011. A Data-Based Approach to Social Influence Maximization. *Proc. VLDB Endow.* 5, 1 (2011), 73–84.

[21] Amit Goyal, Wei Lu, and Laks V. S. Lakshmanan. 2011. CELF++: optimizing the greedy algorithm for influence maximization in social networks. In *WWW*. 47–48.

[22] Qintian Guo, Chen Feng, Fangyuan Zhang, and Sibo Wang. 2023. Efficient Algorithm for Budgeted Adaptive Influence Maximization: An Incremental RR-set Update Approach. *Proc. ACM Manag. Data* 1, 3 (2023), 207:1–207:26.

[23] Qintian Guo, Sibo Wang, Zhewei Wei, and Ming Chen. 2020. Influence Maximization Revisited: Efficient Reverse Reachable Set Generation with Bound Tightened. In *SIGMOD*. 2167–2181.

[24] Qintian Guo, Sibo Wang, Zhewei Wei, Wenqing Lin, and Jing Tang. 2022. Influence Maximization Revisited: Efficient Sampling with Bound Tightened. *ACM Trans. Database Syst.* 47, 3 (2022), 12:1–12:45.

[25] Chris Harshaw, Moran Feldman, Justin Ward, and Amin Karbasi. 2019. Submodular Maximization beyond Non-negativity: Guarantees, Fast Algorithms, and Applications. In *ICML*. 2634–2643.

[26] Keke Huang, Jing Tang, Xiaokui Xiao, Aixin Sun, and Andrew Lim. 2020. Efficient Approximation Algorithms for Adaptive Target Profit Maximization. In *ICDE*. IEEE, 649–660.

[27] Intel. 2024. CPU Metrics Reference. https://www.intel.com/content/www/us/en/docs/vtune-profiler/user-guide/2024-1/cpu-metrics-reference.html.

[28] IZEA. 2023. Average Cost for an Influencer Post: How Much Do Influencers Charge? https://izea.com/resources/average-cost-for-an-influencer-post.

[29] Tianyuan Jin, Yu Yang, Renchi Yang, Jieming Shi, Keke Huang, and Xiaokui Xiao. 2021. Unconstrained Submodular Maximization with Modular Costs: Tight Approximation and Application to Profit Maximization. *Proc. VLDB Endow.* 14, 10 (2021), 1756–1768.

[30] Ehsan Kazemi, Shervin Minaee, Moran Feldman, and Amin Karbasi. 2021. Regularized Submodular Maximization at Scale. In *ICML*, Vol. 139. 5356–5366.

[31] David Kempe, Jon M. Kleinberg, and Éva Tardos. 2003. Maximizing the spread of influence through a social network. In *ACM SIGKDD*. ACM, 137–146.

[32] Yuchen Li, Dongxiang Zhang, and Kian-Lee Tan. 2015. Real-time Targeted Influence Maximization for Online Advertisements. *Proc. VLDB Endow.* 8, 10 (2015), 1070–1081.

[33] LINQIA. 2019. How Much Budget Should You Spend on Influencer Marketing? https://www.linqia.com/insights/how-much-budget-should-you-spend-on-influencer-marketing/.

[34] Bo Liu, Gao Cong, Dong Xu, and Yifeng Zeng. 2012. Time Constrained Influence Maximization in Social Networks. In *ICDM*. 439–448.

[35] Cheng Lu, Wenguo Yang, and Suixiang Gao. 2024. Regularized nonmonotone submodular maximization. *Optimization* 73, 6 (2024), 1739–1765.

[36] Wei Lu and Laks V. S. Lakshmanan. 2012. Profit Maximization over Social Networks. In *ICDM*. 479–488.

[37] Yandong Mao, Eddie Kohler, and Robert Tappan Morris. 2012. Cache craftiness for fast multicore key-value storage. In *EuroSys*. 183–196.

[38] Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, and Amin Karbasi. 2016. Fast Constrained Submodular Maximization: Personalized Data Summarization. In *ICML*.

[39] Hung T. Nguyen, Thang N. Dinh, and My T. Thai. 2016. Cost-aware Targeted Viral Marketing in billion-scale networks. In *INFOCOM*. 1–9.

[40] Christos H. Papadimitriou. 1994. *Computational complexity*. Addison-Wesley.

[41] Georgios Psaropoulos, Ismail Oukid, Thomas Legler, Norman May, and Anastasia Ailamaki. 2019. Bridging the Latency Gap between NVM and DRAM for Latency-bound Operations. 13:1–13:8.

[42] Shopify. 2022. Influencer Pricing: How Much Should You Pay. https://www.shopify.com/blog/influencer-pricing.

[43] Shixuan Sun, Yuhang Chen, Shengliang Lu, Bingsheng He, and Yuchen Li. 2021. ThunderRW: An In-Memory Graph Random Walk Engine. *Proc. VLDB Endow.* 14, 11 (2021), 1992–2005.

[44] Jing Tang, Xueyan Tang, Xiaokui Xiao, and Junsong Yuan. 2018. Online Processing Algorithms for Influence Maximization. In *SIGMOD*.

[45] Jing Tang, Xueyan Tang, and Junsong Yuan. 2016. Profit maximization for viral marketing in Online Social Networks. In *ICNP*. 1–10.

[46] Jing Tang, Xueyan Tang, and Junsong Yuan. 2018. Profit Maximization for Viral Marketing in Online Social Networks: Algorithms and Analysis. *IEEE Trans. Knowl. Data Eng.* 30, 6 (2018), 1095–1108.

[47] Youze Tang, Yanchen Shi, and Xiaokui Xiao. 2015. Influence Maximization in Near-Linear Time: A Martingale Approach. In *SIGMOD*, Timos K. Sellis, Susan B. Davidson, and Zachary G. Ives (Eds.). 1539–1554.

[48] Youze Tang, Xiaokui Xiao, and Yanchen Shi. 2014. Influence maximization: near-optimal time complexity meets practical efficiency. In *SIGMOD*. 75–86.

[49] Upfluence. 2023. How to plan your influencer marketing budget. https://www.upfluence.com/influencer-marketing/how-to-plan-your-influencer-marketing-budget.

[50] David P. Williamson and David B. Shmoys. 2011. *The Design of Approximation Algorithms*. Cambridge University Press.

[51] Laurence A. Wolsey. 1982. Maximising Real-Valued Submodular Functions: Primal and Dual Heuristics for Location Problems. *Math. Oper. Res.* 7, 3 (1982), 410–425.

[52] Yuqing Zhu, Deying Li, Ruidong Yan, Weili Wu, and Yuanjun Bi. 2017. Maximizing the Influence and Profit in Social Networks. *IEEE Trans. Comput. Soc. Syst.* 4, 3 (2017), 54–64.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009