The Most Influenced Community Search on Social Networks

Xueqin Chang[♮], Qing Liu[♮], Yunjun Gao[♮], Baihua Zheng[♯], Yi Cai[‡], Qing Li[†]
[♮]*Zhejiang University,* [♯]*Singapore Management University*
[‡]*South China University of Technology,* [†]*The Hong Kong Polytechnic University*
{changxq, qingliucs, gaoyj}@zju.edu.cn, bhzheng@smu.edu.sg, ycai@scut.edu.cn, csqli@comp.polyu.edu.hk

***Abstract*—In this paper, we address a novel problem in social network analysis: the Most Influenced Community Search (MICS). Given a graph and a seed node set S, the MICS problem seeks to identify a densely connected subgraph that is most significantly impacted by S. We firstly formally define the MICS problem and prove its NP-hardness, demonstrating that no constant-factor approximation is feasible. To efficiently tackle the MICS problem, we propose a framework, consisting of two main phases. The first phase is to compute the influenced expectation for each node, which represents the likelihood of a node being influenced by S. We first develop a sampling-based algorithm, S-InfExp, which effectively estimates the influenced expectations with theoretical guarantee. To enhance the efficiency, we also develop a learning-based approach, L-InfExp, which predicts the influenced expectations more swiftly. In the second phase, we present two algorithms, GlobalSearch and LocalSearch to identify the most influenced community based on the computed influenced expectations. GlobalSearch operates in a top-down approach, greedily removing nodes to obtain a community with higher influenced expectations, while LocalSearch employs a bottom-up strategy, determining the local most influenced community for each node. Experiments on eight real-world datasets show that (1) L-InfExp is up to 2 orders of magnitude faster than S-InfExp while maintaining comparable accuracy, (2) LocalSearch is, on average, 10× faster than GlobalSearch, with both algorithms effectively identifying the community with the highest influenced expectations, and (3) the proposed algorithms outperform all baselines.**

***Index Terms*—Influence Propagation, Community Search, Social Network**

I. Introduction

With the proliferation of social networks, millions of people are now connected and interact with each other on a daily basis. This has sparked significant interest in understanding how influence diffuses across these networks [1]–[4]. Researchers and industry professionals have focused on various aspects of influence analysis, such as predicting influence cascades [1], [5]–[8], identifying influential nodes [3], [9]–[14], and maximizing influence spread [2], [15]–[20].

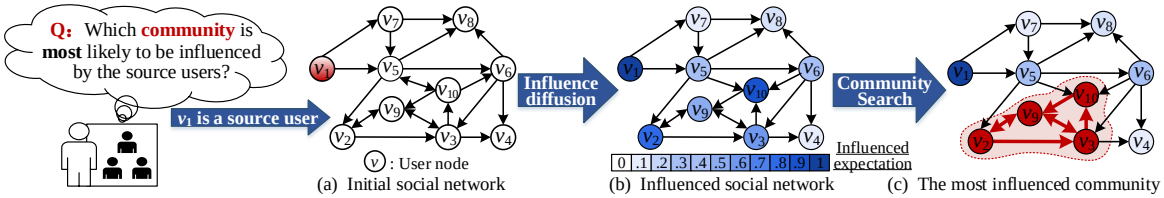
A critical question in this domain is: ***Which users or groups of users are most likely to be influenced by a given set of source users?*** Answering this question is crucial for applications such as targeted marketing [21], epidemic control [22], and personalized recommendation [23]. For example, consider a scenario depicted in Figure 1(a) where a company employs a user v_1 to promote its new product. As v_1's influence propagates through the social network, other users are affected to varying extents, as shown in Figure 1(b). Among them, the highly influenced users $\{v_2, v_3, v_9, v_{10}\}$ form a densely connected subgraph. This subgraph represents a community that is not only the most influenced by the product but also features close-knit structure and frequent interactions, which enhances the likelihood of adoption through group dynamics and shared interests. Targeting such a community can significantly boost marketing effectiveness [21]. We refer to this subgraph as the *most influenced community*. Moreover, in practical applications like marketing and political campaigns, the community needs to be of sufficient size, as larger communities are essential for achieving effective outreach [24].

The Most Influenced Community Search Problem. Based on the above motivation, in this paper, we study the novel Most Influenced Community Search (MICS) problem. Specifically, given a directed graph $G(V_G, E_G)$, a size constraint γ, and a set of seed nodes $S \subset V_G$, our goal is to find a subgraph $H \subseteq G$ with $|H| \geq \gamma$ that is most influenced by S and is densely connected. To quantify the impact of S on a community, we introduce the concept of *influenced expectation*, which represents the likelihood of a node being influenced by S. We define the influenced expectation of a community H as the average[1] of the individual influenced expectations of the nodes within H. Moreover, we employ D-core (a.k.a. (k, l)-core) [26] to assess the structural cohesiveness of the community due to its computation efficiency and broad adoption [27], [28]. The MICS problem has many real-life applications.

Application 1. *During promotional campaigns, companies need to identify the most receptive user groups for targeted marketing [21]. While collaborating with influencers can rapidly promote a new product to a broad audience [29], it is crucial to narrow the marketing scope further to a target group of users who are most likely to engage with the product [30]. Cohesive communities tend to share similar interests and consumption habits, and their strong intra-group bonding can accelerate the spread of product, leading to faster purchasing decisions [31]. By applying MICS, companies can pinpoint the community most affected by the product, where users are not only closely connected to each other, but also have a strong need or interest in the product, thereby optimizing marketing efforts and reducing costs associated with ineffective outreach.*

Application 2. *In managing infectious disease outbreaks, public health authorities like the Centers for Disease Control and Prevention (CDC) must identify high-risk communities for efficient and targeted resource allocation, such as vaccinations and medication distribution [32]. Cohesive communities, such as neighborhoods or workplaces, often share similar lifestyles and health behaviors, facilitating rapid disease spread [22]. MICS can help pinpoint the community most affected by an outbreak, enabling the CDC to optimize prevention and control strategies, thereby reducing the risk of disease transmission.*

[1]The influenced expectation of a graph can be defined using other aggregation functions such as *min*, *max*, and *sum*. For a detailed discussion, please refer to our technical report [25].



(a) Initial social network (b) Influenced social network (c) The most influenced community

Fig. 1. A motivating example

Challenges and Solutions. To effectively and efficiently address the MICS problem, we propose a two-phase framework.

The first phase is to compute the influenced expectation of each node given a seed set S. This involves determining the likelihood that each node will be affected by S. However, calculating the exact influenced expectations is $\#\mathbf{P}$-hard, making it impractical for large graphs. Hence, we turn to approximation methods. Existing techniques in influence maximization, such as Monte-Carlo Simulation [29] and Reverse Influence Sampling (RIS) [16], are designed for a task that is different from ours. These methods estimate the total number of nodes influenced by each node in the social network and are therefore not directly applicable to our problem. Thus, we develop two novel algorithms to calculate influenced expectations for nodes in G, based on a given seed set S.

We firstly propose S-InfExp, which unbiasedly estimates the influenced expectation of each node with theoretical guarantees. Unlike traditional RIS, which estimates a node's influence spread by sampling *reverse reachable sets* from randomly selected nodes, S-InfExp generates an unbiased estimation of influenced expectations by randomly sampling multiple *forward influence propagation instances* from the given seed set S. However, directly traversing the whole graph to generate each instance incurs significant time overhead. To address this challenge, we propose a novel *subset sampling technique*, which models the propagation of each activated node to its out-neighbors as discrete distribution sampling [33]. This approach allows us to skip certain out-neighbors during the generation of each influence propagation instance, improving sampling efficiency by up to $5\times$. Moreover, we develop L-InfExp, a learning-based algorithm to predict the influenced expectation for each node online without generating influence propagation instances. This method involves generating multiple seed sets, computing their corresponding influenced expectations, and using these results to train a nonlinear regression model. L-InfExp achieves up to 2 orders of magnitude faster than S-InfExp, while maintaining comparable accuracy.

The second phase is to identify the most influenced community based on the influenced expectations computed in the first phase. Existing studies [34]–[42] on the influential community search assume that the community either does not exceed a specified size constraint or has no size constraint in their models, which restricts their applicability in the practical scenarios discussed in this paper. To this end, we introduce a new community model, the *most influenced community (MIC)*, which meets or exceeds the size constraint to ensure the community is large enough to produce meaningful and measurable outcomes (e.g., higher adoption rates or broader social impact) in real-world applications [24]. We prove that searching for the *MIC* is **NP**-hard and inapproximable within a constant factor. Thus, directly applying existing enumerative methods [34], [38], [40], [42] to address our problem would be computationally prohibitive. To overcome this challenge, we propose two new algorithms for the second phase.

We firstly propose GlobalSearch. We show that the graph's influenced expectation function is non-monotonic and non-submodular, meaning removing a node from a community D to form another D' can either increase or decrease the influenced expectation. This non-linearity complicates the decision-making process regarding which node to remove to uncover a community with a higher influenced expectation in each iteration. To tackle this challenge, we propose *two heuristic strategies* to greedily remove the nodes within GlobalSearch. Moreover, we present LocalSearch, which sorts nodes by their influenced expectations in descending order. Starting with the node having the largest influenced expectation, LocalSearch identifies the *local most influenced community* containing that node. The process is repeated by removing the node and searching for the local most influenced community for the next highest node. The community with the maximum influenced expectations is ultimately selected. Various pruning techniques are also incorporated to enhance efficiency.

Contributions. Our contributions are summarized as follows.

- We investigate the most influenced community search problem in social networks for the first time.
- We propose a sampling-based algorithm, S-InfExp, to estimate the influenced expectations with theoretical guarantees. Additionally, we develop a learning-based method, L-InfExp, for efficient prediction of influenced expectations.
- We design GlobalSearch and LocalSearch algorithms to identify the most influenced community, incorporating various pruning strategies to improve search efficiency.
- We conduct extensive experiments on eight real networks to demonstrate the effectiveness, efficiency, and scalability of the proposed algorithms.

Roadmap. Section II reviews the related work. Section III defines and analyzes the problem of MICS. Section IV provides an overview of proposed techniques. Section V presents S-InfExp and L-InfExp algorithms. Section VI introduces GlobalSearch and LocalSearch algorithms. Section VII reports experimental result and Section VIII concludes the paper.

II. Related Work

Community Search. Community search aims to find cohesive subgraphs containing given query nodes [43]. It has been widely studied across different graph types, including directed graphs [26], [27], [44], attributed graphs [45]–[47], and heterogeneous graphs [48], [49]. Recently, learning-based algorithms have been proposed for community search [50], [51].

The most closely related community search to our work is influential community search [34]–[42], which aims to find top-r communities with the highest influence scores. However, influential community search differs from our problem in two key aspects. First, in influential community search, the nodes' influence scores are typically given in advance and are fixed. In contrast, our problem requires computing the influenced expectation for each node under a given set of seed nodes, as the influenced expectations vary with different input seed node sets. Second, our problem and the influential community search impose different constraints on the community size, i.e., our problem specifies a lower bound, while influential community search specifies an upper bound or no bound. Therefore, techniques designed for influential community search cannot be directly applied to our problem.

Community Detection. Community detection aims to retrieve all communities that meet specific constraints [52]–[54]. Recent studies have employed clustering techniques for community detection [55]–[57]. However, these works did not require structural or size constraints of community answers, nor do they consider the effects of influence propagation, which is the focus of our MICS problem in this paper.

Influence Analysis. Influence analysis has been extensively studied in social networks and includes topics such as influence cascade prediction [1], [5]–[8], influential nodes identification [3], [9]–[14], and influence maximization [2], [15]–[20], [58], [59]. Specifically, influence cascade prediction forecasts diffusion volume, identifies future active nodes, and clarifies propagation relationships among nodes. Influential nodes identification focuses on finding the top-k most influential nodes. Influence maximization aims to identify a set of k seed nodes to maximize the expected number of influenced nodes. In this paper, we introduce a new topic in influence analysis, finding the most influenced community for a given seed node set, offering a distinct perspective compared to existing methods.

III. Preliminaries

In this section, we formally define and analyze the most influenced community search (MICS) problem.

A. Problem Formulation

We consider a directed graph $G(V_G, E_G)$, where V_G and E_G represent the sets of n nodes and m edges, respectively. Each directed edge $e = (u, v) \in E_G$ is associated with an influence weight $w(u, v) \in [0, 1]$, which quantifies the influence of node u on node v. In this context, u is referred to as an in-neighbor of v, and v is an out-neighbor of u. For a node $v \in V_G$, let $N_G^{in}(v)$ (resp. $N_G^{out}(v)$) denote the set of in-neighbors (resp.out-neighbors) of v. Correspondingly, the in-degree of v, denoted by $d_G^{in}(v) = |N_G^{in}(v)|$, represents the number of in-neighbors of v in G. Similarly, the out-degree of v, denoted by $d_G^{out}(v) = |N_G^{out}(v)|$, represents the number of out-neighbors of v in G. To define the MICS problem, we introduce the cohesive subgraph model and influence propagation model.

Cohesive subgraph model. In this paper, we employ the D-core (a.k.a. (k, l)-core) to measure community structure density due to its computation efficiency and broad adoption [28].

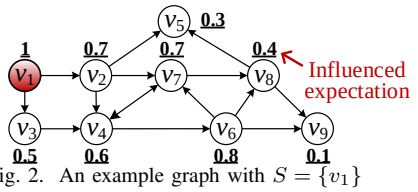


Fig. 2. An example graph with $S = \{v_1\}$

Definition 1. (D-core) [26]. *Given a directed graph $G(V_G, E_G)$, two integers k and l, a subgraph $H(V_H, E_H) \subseteq G$ is a D-core of G, if it satisfies the following conditions:*

1) ***Cohesive:*** *$\forall v \in V_H$, $d_H^{in}(v) \geq k$ and $d_H^{out}(v) \geq l$.*
2) ***Maximal:*** *For any subgraph $H' \subseteq G \wedge H' \supset H$ that contains H, H' is not a D-core.*

In other words, a D-core is a maximal subgraph of G, in which each vertex's in-degree and out-degree are at least k and l, respectively. For example, in Figure 2, the directed subgraph H_1 induced by nodes v_4, v_6, and v_7 is a (1,1)-core because $\forall v \in V_{H_1}$, $d_{H_1}^{in}(v) = d_{H_1}^{out}(v) = 1$, and we cannot reach another D-core by including any other vertex to H_1.

Influence propagation model. We employ the Independent Cascade (IC) model [29] to simulate influence propagation, as it is widely adopted to depict the influence propagation process [15], [20]. *It is worth noting that our problem and solutions can be extended to other models.* In the IC model, for a given seed node set $S \subset V_G$, the influence propagation process unfolds in discrete steps as follows. At step 0, all nodes in S are activated, while the remaining nodes remain inactive. Once a node is activated, it stays activated throughout the propagation process. In each subsequent step, if a node u is activated at step i, it has a single chance to activate each of its inactive out-neighbors $v \in N_G^{out}(u)$ with a probability determined by the influence weight $w(u, v)$ at step $i + 1$. After this attempt, u cannot activate any additional nodes. The process continues until no more nodes can be activated.

Based on the influence propagation model, for each node $v \in V_G$, we define the influenced expectation as follows.

Definition 2. (Influenced expectation). *Given a directed graph $G(V_G, E_G)$ and a seed node set $S \subset V_G$, for each node $v \in V_G$, we define $Ie(v, S)$ as the influenced expectation of v, which represents the probability that v will be activated by S under the IC model.*

For instance, in Figure 2, v_1 is the seed node (i.e., $S = \{v_1\}$), and numbers next to vertices represent their influenced expectations under the current seed node set S. For example, $Ie(v_8, \{v_1\}) = 0.4$ indicates that there is a 40% chance of v_8 being activated by v_1. For a subgraph $H(V_H, E_H) \subseteq G$, we define the influenced expectation of H as follows.

Definition 3. (Influenced expectation of H). *Given a directed graph G, a subgraph $H(V_H, E_H) \subseteq G$, and a seed node set $S \subset V_G$, the influenced expectation of H, denoted by $Ie(H, S)$, is the average expectation that H is influenced by S under the IC model. Specifically,*

$$Ie(H, S) = \frac{\sum_{v \in V_H} Ie(v, S)}{|V_H|}.$$

In Definition 3, $Ie(H, S)$ is defined as the average of the influenced expectations of all nodes in H. Note that $Ie(H, S)$

could also be defined using other aggregation functions, such as *min*, *max*, or *sum*. Extensions to these functions are discussed in our technical report [25].

Based on the above definitions, we introduce a novel community model called the most influenced community (MIC).

Definition 4. (MIC). *Given a directed graph $G(V_G, E_G)$, a seed node set $S \subset V_G$, and integers k and l, a subgraph $H(V_H, E_H) \subseteq G$ is an MIC if it satisfies:*

1) ***Cohesive:*** *H is a D-core, i.e., $\forall v \in V_H$, $d_H^{in}(v) \geq k$ and $d_H^{out}(v) \geq l$.*
2) ***Connected:*** *H is a connected subgraph.*
3) ***Most influenced:*** *H has the maximum influenced expectation $Ie(H, S)$ among all the subgraphs that satisfy above two conditions.*
4) ***Maximal:*** *There is no other subgraph $H' \subseteq G \wedge H' \supset H$ that satisfies conditions 1), 2), and 3).*

In summary, the MIC is a D-core with the highest influenced expectation. Then, we formally define the MICS problem.

Problem 1. (MICS). *Given a directed graph $G(V_G, E_G)$, a seed node set $S \subset V_G$, two integers k and l, and a size constraint γ, the objective of MICS problem is to find the MIC with $|MIC| \geq \gamma$.*

The size constraint γ ensures that the community is large enough for practical applications, as discussed in Section I.

Example 1. *Figure 2 shows a directed graph G. Let $S = \{v_1\}$, $\gamma = 2$, $k = 1$, and $l = 1$. The number next to each node is its influenced expectation w.r.t., S. The directed subgraph H_1 induced by nodes v_4, v_6, and v_7 is a (1, 1)-core and has the maximum influenced expectation $Ie(H_1, \{v_1\}) = (0.6 + 0.8 + 0.7)/3 = 0.7$. Hence, H_1 is the result of MICS.*

B. Problem Analyses

In this section, we demonstrate the NP-hardness of the MICS problem by breaking it down into two sub-problems.

Problem 2. (IEC). *Given a directed graph $G(V_G, E_G)$ and a seed node set $S \subset V_G$, the problem of Influenced Expectation Computation (IEC) is to compute the influenced expectation $Ie(v, S)$ for every node $v \in V_G$.*

Problem 3. (CS). *Given a directed graph $G(V_G, E_G)$, two integers k and l, a size constraint γ, and the influenced expectation $Ie(v, S)$ for every node $v \in V_G$, the problem of Community Search (CS) is to find the MIC with $|MIC| \geq \gamma$.*

For Problems IEC and CS, we have the following theorems.

Theorem 1. *The IEC is #P-hard under the IC model.*

Proof. We prove this theorem by a reduction from the Probabilistic s-t Connectivity (PC) problem in a directed graph [60]. Specifically, give a directed graph $G(V_G, E_G)$, where each edge has an existence probability of 0.5, and two nodes $s \in V_G$ and $t \in V_G$, the PC problem is to compute the probability that s is connected to t. According the PC problem, we set up the IEC problem on G as follows: (1) the influence weights of all edges are set to 0.5; (2) $S = \{s\}$; and (3) $v = t$. Clearly, t is influenced by s if and only if there is a directed path from s to t. Thus, the probability that s is connected to t equals $Ie(t, s)$. Since the PC problem is #P-complete [60], the IEC problem is also #P-hard. □

Theorem 1 shows the hardness of computing exact influenced expectation. Therefore, in Section V, we develop approximation algorithms to estimate the influenced expectation $Ie(v, S)$ for each node $v \in V_G$ w.r.t. a given set of seed nodes S. Assuming we have computed the influenced expectation of each node, we show that searching the MIC is NP-hard with no constant-factor approximation.

Theorem 2. *The CS problem is NP-hard.*

Proof. We prove this by reducing the Decision of the Maximum Clique (DMC) problem to the CS problem. Specifically, given an undirected graph $G'(V', E')$ and an integer k', the DMC is to determine whether G' contains a k'-clique. Based on the DMC, we construct a directed graph $G(V, E)$ for the CS problem as follows: (1) $V = V'$; (2) $\forall u, v \in V'$, if there exists an edge between u and v in E', add both (u, v) and (v, u) to E; (3) add a new vertex $w \notin V'$ to V, and $\forall v \in V'$, add both (w, v) and (v, w) to E; (4) set $Ie(w, S)$ to a positive integer, while for the remaining vertices in V, $Ie(v, S)$ is set to 0. We set the parameters of the CS problem as: $k = l = \gamma = k' + 1$. If there exists a polynomial-time algorithm for the CS problem, we could solve the DMC. Specifically, let H be the community returned by the CS problem. Then, the influenced expectation of H is $Ie(H, S) = \frac{Ie(w,S)}{|H|}$. To maximize $Ie(H, S)$, $|H|$ should be minimized. The smallest possible $|H|$ is $k'+1$, which forms a D-core with bidirectional edges between any pair of nodes. If $|H| = k' + 1$, the subgraph of G' induced by $\{V_H - w\}$ is a k'-clique. Otherwise, G' does not contain a k'-clique. This contradicts the fact that the DMC problem is NP-complete. Thus, the CS problem is NP-hard. □

Theorem 3. *There are no constant-factor approximation algorithms for the CS problem.*

Proof. As demonstrated by Amini et al. [61], for $r \geq 3$, the problem of finding the minimum subgraph where each vertex's degree is at least r (SMSMD$_r$) does not exist any constant-factor approximation, unless $P = NP$. As shown in Theorem 2, the CS problem requires minimizing $|H|$. If we set $k = l = r$ and $\gamma = 0$, the CS problem is equivalent to SMSMD$_r$ problem. Consequently, there are no constant-factor approximation algorithms for the CS problem. □

Based on Theorems 1, 2, and 3, we can conclude that the MICS problem is NP-hard.

Theorem 4. *The MICS problem is NP-hard, and no constant-factor approximation algorithms exist for it.*

Next, we analyze the objective function $Ie(H, S)$ for the MICS problem, where H is a D-core of size at least γ.

Theorem 5. *$Ie(H, S)$ is non-monotonic and non-submodular*[2].

Due to the non-monotonicity and non-submodularity of $Ie(H, S)$, existing greedy and other approximation algorithms designed for monotonic and submodular objective functions are not applicable to our problem. Therefore, new algorithms are necessary for solving the MICS problem.

[2]Due to the space limitation, all the omitted proofs are moved to [25].

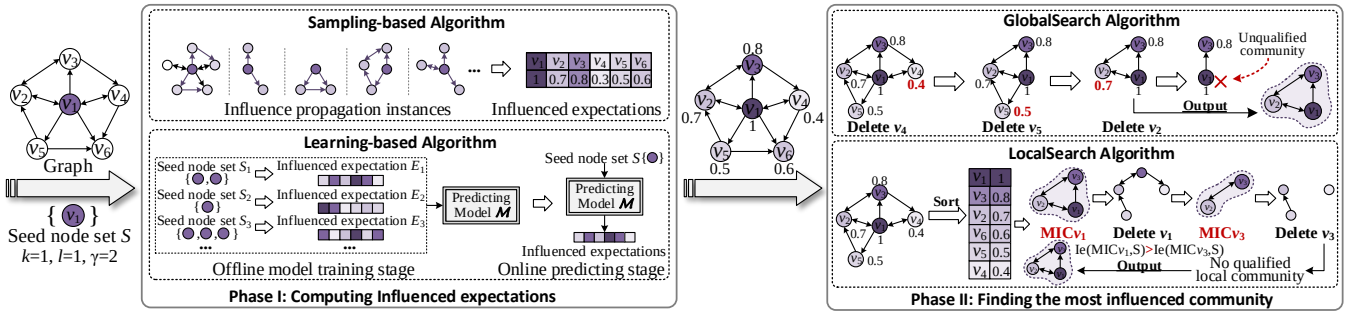


Fig. 3. Framework overview

IV. Framework Overview

To address the MICS problem, we propose a framework illustrated in Figure 3, which comprises two main phases:

- Phase I involves computing the influenced expectation for each node, a task proven to be #P-hard as shown in Theorem 1. To address this, we present two novel algorithms. S-InfExp is to randomly sample a sufficient number of forward influence propagation instances to unbiasedly estimate the influenced expectation of each node. Based on the influenced expectations obtained from S-InfExp, L-InfExp applies nonlinear regression models to predict the influenced expectation for each node, which can significantly improve the efficiency.
- Phase II focuses on retrieving the most influenced community that satisfies specific degree and size constraints. Given the NP-hardness of the CS problem, we design two new algorithms. GlobalSearch finds the community in a top-down manner by iteratively removing the node with the smallest influenced expectation. In contrast, LocalSearch identifies the community in a bottom-up manner by finding the community containing node v that has the maximal influenced expectation for each $v \in V_G$.

In the subsequent two sections, we will provide detailed descriptions of the algorithms used in Phases I and II.

V. Influenced Expectation Computation

In this section, we propose a sampling-based algorithm S-InfExp to approximate the influenced expectation of each node with theoretical guarantee, followed by a learning-based method L-InfExp for predicting these expectations.

A. Sampling-based Algorithm

In Section III-A, we introduced the IC model to simulate influence propagation. Building on this, we define the concept of influence propagation instance as follows.

Definition 5. (Influence propagation instance). *Given a directed graph $G(V_G, E_G)$, a seed node set $S \subset V_G$, and a subgraph $g \subseteq G$, g is considered an influence propagation instance w.r.t. S if it satisfies: (1) g contains S, and (2) for every vertex $v \in V_g - S$, there exists a path from a seed node $s \in S$ to v within g.*

Intuitively, for a given seed node set S, an influence propagation instance g represents a scenario where all vertices in g are activated by S, while vertices outside g are not activated. We denote (1) $p(g)$ as the existence probability of g, and (2) $\mathcal{G}$ as the set of all influence propagation instances w.r.t. S. Obviously, $\sum_{g\in\mathcal{G}} p(g) = 1$. Recall that $Ie(v, S)$ represents the expectation of v being activated by S. Based on the influence propagation instance, $Ie(v, S)$ can be computed as follows.

$$Ie(v, S) = \sum\nolimits_{g\in\mathcal{G}\wedge v\in V_g} p(g)$$

In other words, $Ie(v, S)$ is the sum of the existence probabilities of the influence propagation instances that include v. Computing $Ie(v, S)$ directly by enumerating all influence propagation instances is computationally prohibitive. Instead, we use sampling technique to sample partial influence propagation instances to estimate $Ie(v, S)$. This involves addressing three main issues: (1) How to estimate $Ie(v, S)$ from sampled influence propagation instances, (2) How to sample influence propagation instances, and (3) How to ensure the accuracy of the estimated $Ie(v, S)$.

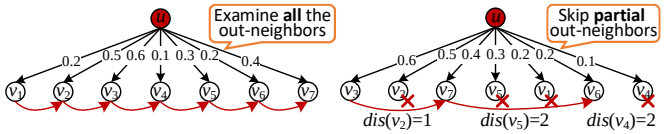
Estimating $Ie(v, S)$ from Sampled Influence Propagation Instances. Given a set of independent influence propagation instances $\mathcal{I} = \{I_1, I_2, \cdots\}$ originating from the seed node set S on graph $G(V_G, E_G)$, the estimate of v's influenced expectation, denoted by $\tilde{Ie}(v, S)$, is given by

$$\tilde{Ie}(v, S) = \frac{|\{I | I \in \mathcal{I} \wedge v \in V_I\}|}{|\mathcal{I}|}.$$

In essence, we employ the proportion of sampled instances in $\mathcal{I}$ that include v as an estimate for $Ie(v, S)$.

Sampling Influence Propagation Instances. One straightforward method is to traverse the graph according to the IC model. Starting from the seeds, we activate them and then explore their out-neighbors. For each unactivated out-neighbor v of an activated node u, we generate a random number between 0 and 1. If this number exceeds the influence weight $w(u, v)$, v is activated, as shown in Figure 4(a). This process continues until no further vertices can be activated. The subgraph induced by the activated vertices forms an influence propagation instance. Generating each influence propagation instance takes $O(\sum_{u\in V_G} d_G^{out}(u)) = O(|E_G|)$ time, and sampling a set of instances $\mathcal{I}$ requires $O(|\mathcal{I}| \cdot |E_G|)$ time.

To enhance efficiency, we propose a *subset sampling technique*, where the propagation of each activated node to its out-neighbors is modeled as discrete distribution sampling [33] during the generation of each influence propagation instance. Specifically, for an activated node u with t out-neighbors v_1, $\cdots$, v_t where $t = d_G^{out}(u)$, we first sort the out-neighbors by their influence weights $w(u, v)$ in descending order. We then select a node to activate in two steps. (1) We choose the first



(a) Original sampling method (b) Subset sampling method

Fig. 4. Illustrations of sampling methods

node v' as the pivotal node and use $w(u, v')$ to determine a skip distance $dis(v')$ based on a geometric distribution $G(w(u, v'))$, where $dis(v') = \left\lfloor \frac{\log(rand())}{\log(1-w(u,v'))} \right\rfloor$ denotes the number of nodes skipped during the sampling. (2) We select the $dis(v')$-th node after v', denoted v''. To ensure v'' is activated with probability $w(u, v'')$, we sample a random number $R \in [0, 1)$ and activate v'' if $R \leq \frac{w(u,v'')}{w(u,v')}$. Then, the node after v'' is set as the pivotal node for the next node selection. This process continues until no more nodes v'' can be selected. According to [33], for a node u, subset sampling for discrete distributions achieves an expected time complexity of $O(1 + \kappa + \log d_G^{out}(u))$, where $\kappa = \sum_{v \in N_G^{out}(u)} w(u, v)$. Figure 4(b) illustrates subset sampling method, showing the skipped vertices v_2, v_5, v_1, and v_4.

Ensuring Estimate Quality. Intuitively, a larger number of samples lead to higher quality of the estimated influenced expectations, albeit at a higher cost. To ensure that $\tilde{Ie}(v, S)$ is an unbiased estimate of $Ie(v, S)$, we extend *chernoff inequalities* [62] to analyze the required sample size $|\mathcal{I}|$.

Theorem 6. *Given a directed graph $G(V_G, E_G)$ and a seed node set $S \subset G$, for any node $v \in V_G$, the estimate $\tilde{Ie}(v, S)$ derived via subset sampling satisfies $|\tilde{Ie}(v, S) - Ie(v, S)| < \epsilon \cdot Ie(v, S)$ with probability of at least $(1 - n^{-l})$ when $|\mathcal{I}| \geq \frac{l \cdot (2+\epsilon) \cdot \log n}{\epsilon^2 \cdot Ie(v,S)}$, where $n = |V_G|$, $\epsilon \in (0, 1)$ is a user-specified error parameter, and l is chosen to ensure a success probability of $1 - \frac{1}{n}$.*

Note that Theorem 6 offers a theoretical bound on the required sample size. By the law of large numbers, in practice, as $|\mathcal{I}|$ increases, $\tilde{Ie}(v, S)$ converges to $Ie(v, S)$.

S-InfExp Algorithm. After addressing the three key issues mentioned above, we propose a sampling-based algorithm, S-InfExp, to compute the influenced expectation for every node. Algorithm 1 provides the pseudo code. Specifically, S-InfExp begins by initializing $I(v)$ for each node v, which tracks the number of influence propagation instances containing v (Line 1). The algorithm then generates α influence propagation instances (Lines 3-21). For each instance, S-InfExp initializes an empty queue Q and marks all nodes as unactivated (Line 4). All the seed nodes are activated and added to Q (Lines 5-7). S-InfExp performs a BFS traversal of the activated nodes. For each activated node u, S-InfExp sorts the out-neighbors of u in descending order of their influence weights and sets the first node as the pivotal node (Lines 9-11). Then, S-InfExp utilizes the subset sampling technique to select out-neighbors of u to activate (Lines 13-20). If a node v is activated, $I(v'')$ is updated correspondingly (Line 19). After generating α influence propagation instances, S-InfExp computes the estimated influenced expectation $\tilde{Ie}(v, S)$ for each node (Lines 22-23). Finally, the algorithm returns the estimated influenced expectations of all nodes (Line 24).

Algorithm 1 S-InfExp

Input: a graph $G(V_G, E_G)$, a seed set S, and a sample size α
Output: $\tilde{Ie}(v, S)$ for $\forall v \in V_G$

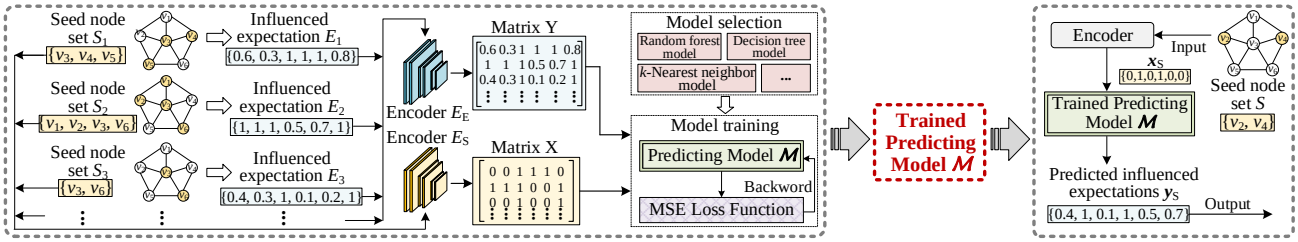
1: **for** each $v \in V_G$ **do**
2: $I(v) \leftarrow 0$;
3: **for** $j \leftarrow 1$ to α **do**
4: Queue $Q \leftarrow \emptyset$; Mark $\forall v \in V_G$ as unactivated;
5: **for** each $s \in S$ **do**
6: Mark s as activated; Add s to Q;
7: $I(s) \leftarrow I(s) + 1$;
8: **while** $Q \neq \emptyset$ **do**
9: Pop u from Q;
10: Sort u's out-neighbors by descending influence weight;
11: $v' \leftarrow$ The first out-neighbor of u;
12: $i \leftarrow 0$;
13: **while** $i < d_G^{out}(u)$ **do**
14: $dist(v') \leftarrow \lfloor log(rand())/log(1 - w(u, v')) \rfloor$;
15: $i \leftarrow i + dist(v')$;
16: **if** $i \geq d_G^{out}(u)$ **then break**;
17: $v'' \leftarrow$ the $dist(v')$-th out-neighbor after v;
18: **if** $rand() \leq w(u, v'')/w(u, v') \wedge v''$ is inactive **then**
19: $I(v'') \leftarrow I(v'') + 1$;
20: Insert v'' into Q; Mark v'' as activated;
21: $v' \leftarrow$ the node after v''; $i++$;
22: **for** each $v \in V_G$ **do**
23: $\tilde{Ie}(v, S) \leftarrow I(v)/\alpha$;
24: **Return** $\cup_{v \in V_G} \tilde{Ie}(v, S)$.

Time Complexity. The expected time cost for generating an influence propagation instance is $O(\sum_{u \in V_G} \log d_G^{out}(u)) = O(|V_G| \log(|V_G|))$, as in the worst case, the out-degree of each node can approach $|V_G - 1|$. Thus, the overall time complexity of S-InfExp is $O(\alpha \cdot |V_G| \log(|V_G|))$.

B. Learning-based Algorithm

While S-InfExp is effective for computing influenced expectations, its time complexity can be prohibitive for real-time analysis due to the need for multiple graph traversals. To address this, we propose a learning-based algorithm, L-InfExp, which predicts the influenced expectation for each node without requiring generating influence propagation instances. L-InfExp operates on the following principle: Given a graph G, we randomly select seed node sets and use S-InfExp to compute nodes' influenced expectations for these seed sets. These seed node sets and their corresponding influenced expectations are then used as training data to build a predictive model. Once trained, this model can quickly predict the influenced expectations of nodes for new seed node sets, thereby dramatically reducing the influenced expectation computation time and thus enabling online analysis.

The workflow of L-InfExp is illustrated in Figure 5, which consists of two stages, offline model training and online prediction of influenced expectation. In the offline model training stage, we generate training data, which includes seed node sets and their corresponding influenced expectations. Using this data, we train a predictive model to estimate influenced expectations. This stage produces a trained model for future predictions. In the online predicting stage, for a given seed set S, we encode S into a multi-hot vector and input it to the trained model to predict the influenced expectation for each node. Next, we present a few key components of L-InfExp.



(a) Offline model training stage (b) Online predicting stage

Fig. 5. The workflow of L-InfExp

Model selection. For predicting influenced expectations, we use nonlinear regression techniques. Specifically, nonlinear regression is a type of regression analysis to model the complex relationships between dependent and independent variables. Below is the general form of a nonlinear regression model,

$$y = f(x_1, x_2, \cdots, x_n) + \epsilon,$$

where y is the dependent variable, x_is are independent variables, f is a nonlinear function, and ϵ represents the error term. In our case,

$$Ie(v, S) = f_v(x_{v_1}, x_{v_1}, \cdots, x_{v_n}) + \epsilon,$$

where x_{v_i} indicates whether node v_i is in the given seed node set S, and f_v predicts $Ie(v, S)$ for node v. Hence, our task is to train f_v for each node v.

Various models can be used for nonlinear regression, such as Random Forest [63], Decision Tree [64], and k-Nearest Neighbor models [65]. It is worth mentioning that L-InfExp can adopt any of the aforementioned models. We will evaluate and compare these models in our experiments. For simplicity, we denote the predictive model as $\mathcal{M}$.

Encoders. The training data consists of two parts: the seed node sets $\mathcal{S}$ and their corresponding influenced expectations sets $\mathcal{E}$. Each seed node set $S_i \in \mathcal{S}$ is encoded as a multi-hot vector $\boldsymbol{x}_i \in \{0,1\}^n$ where $n = |V_G|$. If $v_j \in S_i$, $\boldsymbol{x}_i[j] = 1$. Otherwise $\boldsymbol{x}_i[j] = 0$. Take the graph G with $|V_G| = 11$ plotted in Figure 2 as an example. For the seed node set $S = \{v_3, v_5, v_{10}\}$, the encoded vector is $\boldsymbol{x} = [0,0,0,1,0,1,0,0,0,0,1]$. In addition, each influenced expectations set $E_i \in \mathcal{E}$ w.r.t., S_i is encoded as a probability vector $\boldsymbol{y}_i \in [0,1]^n$, where $\boldsymbol{y}_i[j]$ represents the influenced expectation for node v_j w.r.t. S_i. Let $|\mathcal{S}| = |\mathcal{E}| = \beta$. We encode all seed nodes sets $\mathcal{S}$ and their corresponding influenced expectations $\mathcal{E}$ into multi-hot matrices X and Y, respectively, where

$$X = [\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_\beta]^\top, Y = [\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_\beta]^\top.$$

Then, matrices X and Y are employed to train $\mathcal{M}$ by minimizing the loss function.

Loss function. We adopt the Mean Squared Error (MSE) to assess the model's performance: minimize the MSE between the predicted influenced expectations $\hat{\boldsymbol{y}}_i \in [0,1]^n$ and the ground-truth influenced expectations $\boldsymbol{y}_i \in [0,1]^n$ for each seed node set $S_i \in \mathcal{S}$, where $\hat{\boldsymbol{y}}_i[j]$ is the predicted influenced expectation of node v_j, and $\boldsymbol{y}_i[j]$ denotes the ground-truth influenced expectation of node v_j. The loss function can be expressed as follows:

$$\min \mathcal{L} = \frac{1}{\beta} \sum\nolimits_{i=1}^{\beta} \|\hat{\boldsymbol{y}}_i - \boldsymbol{y}_i\|_2$$

Online prediction. During the online influenced expectation prediction stage, we encode the given seed node set S into a multi-hot vector $\boldsymbol{x}_S \in \{0,1\}^n$, which is then input into the trained model $\mathcal{M}$ to obtain the vector $\boldsymbol{y}_S$, which represents the predicted influenced expectations for all nodes.

VI. MIC Search

After computing the influenced expectation of each node for a given seed set, in this section, we propose GlobalSearch and LocalSearch to identify the most influenced community.

A. Global Search Algorithm

Our goal is to identify the most influenced community with a size of at least γ, i.e., the D-core D that maximizes the influenced expectation while meeting the size constraint γ. Finding a D-core is straightforward by iteratively removing vertices that do not satisfy the in-degree and out-degree constraints. However, the challenge lies in finding the D-core with the maximum influenced expectation. As demonstrated in Theorem 5, the function $Ie(D, S)$ is non-monotonic and non-submodular, meaning that removing a vertex from D to form another D-core D' can either increase or decrease $Ie(D', S)$. Hence, we need to check all possible D-cores contained within D to find the one with the maximum influenced expectation. Clearly, enumerating all such D-cores for a given D is computationally expensive. Thus, we propose two greedy strategies to find the D-core with the highest influenced expectation.

Specifically, there are two potential greedy strategies. For a given connected D-core D, assume we remove a vertex v from D to obtain a new connected D-core D'_v. The first greedy strategy, denoted by Greedy I, is to delete v with the minimum influenced expectation, i.e., $v = \arg\min_{u \in V_D} Ie(u, S)$. The second greedy strategy, denoted by Greedy II, is to remove v such that the influenced expectation of D'_v is maximized, i.e., $v = \arg\max_{u \in V_D} Ie(D'_u, S)$ where $D'_u = D - \{u\}$. To implement Greedy II, we must delete each vertex and check whether D'_v yields the maximum influenced expectation. We compare the performance of these two strategies on eight graphs. We observe that (1) in Figure 6(a), the D-cores returned by Greedy I and Greedy II have similar influenced expectations, and (2) in Figure 6(b), Greedy I is up to 2 orders of magnitude faster than Greedy II. Therefore, we employ Greedy I in our algorithm. Next, we introduce a theorem to prune some D-cores that cannot contain the D-core with the maximum influenced expectation.

Theorem 7. *Given two D-cores D_1 and D_2, and another D-core D_3 contained within D_2, i.e., $D_3 \subseteq D_2$, if $Ie(D_1, S) > \max_{v \in V_{D_2}} Ie(v, S)$, then $Ie(D_1, S) > Ie(D_3, S)$.*

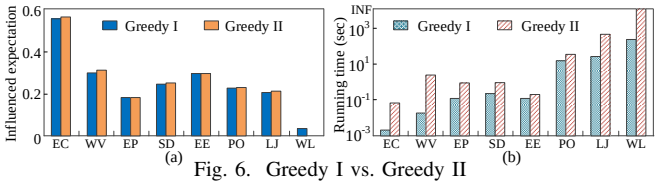


Fig. 6. Greedy I vs. Greedy II

Theorem 7 indicates that if the upper bound of D-core's influenced expectation (i.e., $\max_{v \in V_D} Ie(v, S)$) is lower than that of the currently found best community, then the corresponding D-core definitely does not contain the result community we are looking for and hence can be safely pruned.

Based on these considerations, we propose the algorithm GlobalSearch, with its pseudo code outlined in Algorithm 2. Initially, GlobalSearch performs initialization (Line 1) and computes the maximal D-core D of the input graph G (Line 2). Each connected component D_i of D with $|D_i| \geq \gamma$ is checked to determine if it may contain the D-core with the maximum influenced expectation and is added to the queue Q for further examination (Lines 3-7). If Q is not empty, GlobalSearch pops a D-core from Q and applies Greedy I to obtain a new D-core D (Lines 9-12). Similarly, GlobalSearch checks each connected component of D (Lines 9-17). When no D-core remains to be checked (i.e., $Q = \emptyset$), GlobalSearch returns the most influenced community (Line 18).

Time Complexity. The time complexity for computing the D-core and traversing all connected components in the D-core is $O(|V_G| + |E_G|)$. In the worst case, we may need $|V_G|$ iterations to traverse all D-cores and their connected components, resulting in a time complexity of $O(|V_G| \cdot (|V_G| + |E_G|)$. Thus, the time complexity of Algorithm 2 is $O(|V_G| \cdot (|V_G| + |E_G|)$.

B. Local Search Algorithm

The GlobalSearch algorithm finds the most influenced community using a top-down approach, iteratively removing vertices with the minimum influenced expectation from D-cores. In this subsection, we propose a new bottom-up approach to discover the most influenced community.

Peng et al [40] proposed a local algorithm for influential community search using the *Avg* function. The basic idea is to check the nodes in the maximal k-core one by one, and progressively add vertices to the community according to the descending order of the influence scores of their neighboring nodes, and finally output the top-r communities with the largest influence scores. We extend this local algorithm to apply to our problem, each time a vertex is added to the community, we evaluate whether the updated community still meets the size constraint γ and whether it contains a D-core. If both conditions are met, this D-core is returned as the result. The community with the largest influenced expectation is finally served as the MIC. However, this method is less efficient and often results in a community with suboptimal influenced expectation, as demonstrated by the experimental results in Section VII. To address this, we propose a new local algorithm, LocalSearch.

The core idea behind our LocalSearch is to identify the *local most influenced community* for each node and ultimately return the one with the maximum influenced expectation. Here, the local most influenced community denotes the D-core that has at least γ size, contains a specific node, and has the maximal influenced expectation. Specifically, LocalSearch first computes the maximal D-core D of the given graph and sorts all nodes of D in descending order of their influenced expectations. Then, starting from the node v with the maximum influenced expectation, LocalSearch finds a new D-core $D' \subseteq D$ with the maximal influenced expectation containing v as the local most influenced community, denoted as MIC_v. After MIC_v is found, LocalSearch removes v from D, and repeats the process to find $\text{MIC}_{v'}$ for the next node v'. This continues until all nodes are processed. Finally, LocalSearch returns the local most influenced community with the maximum influenced expectation.

Algorithm 2 GlobalSearch

Input: a graph $G(V_G, E_G)$, two parameters k and l, a size constraint γ, the influenced expectation $Ie(v, S)$ for $\forall v \in V_G$
Output: the most influenced community MIC with $|MIC| \geq \gamma$

1: $MIC \leftarrow \emptyset$; Queue $Q \leftarrow \emptyset$;
2: Compute the maximal D-core D of G;
3: **for** each connected component D_i of D with $|D_i| \geq \gamma$ **do**
4: **if** $Ie(D_i, S) \geq Ie(MIC, S)$ **then**
5: $MIC \leftarrow D_i$;
6: **if** $\max_{v \in V_{D_i}} Ie(v, S) \geq Ie(MIC, S)$ **then**
7: Add D_i into Q;
8: **while** $Q \neq \emptyset$ **do**
9: Pop a D-core D from Q;
10: $v \leftarrow \arg\min_{u \in V_D} Ie(u, S)$;
11: Remove v from D;
12: Maintain D by iteratively removing vertices that do not satisfy degree constraints of D-core;
13: **for** each connected component D_i of D with $|D_i| \geq \gamma$ **do**
14: **if** $Ie(D_i, S) \geq Ie(MIC, S)$ **then**
15: $MIC \leftarrow D_i$;
16: **if** $\max_{v \in V_{D_i}} Ie(v, S) \geq Ie(MIC, S)$ **then**
17: Add D_i into Q;
18: **Return** MIC.

A key challenge for LocalSearch is efficiently computing the local most influenced community MIC_v for each node v. This involves gradually expanding v's neighbors. Specifically, MIC_v is initialized with v, and then the i-hop neighbors of v are added to MIC_v sequentially for $i = 1$ to $i = diam(G)$, where $diam(G)$ denotes the diameter of the graph G. Among neighbors with the same hop distance from v, those with higher influenced expectations are prioritized. After each addition, MIC_v is examined to determine if it contains a D-core that includes node v and meets the size constraint γ. If such a D-core exists, it is returned as MIC_v.

To enhance efficiency, we introduce several pruning strategies in LocalSearch to avoid unnecessary computations.

Pruning strategy 1. Assume nodes are processed in descending order of their influenced expectations. Let MIC_{v_i} represent the local most influenced community for node v_i. For the subsequent node v_{i+1}, if $Ie(\text{MIC}_{v_i}, S) > Ie(v_{i+1}, S)$, then we do not need to compute the local most influenced communities for nodes after v_i. This is because MIC_{v_i} can only be formed by vertex v_i and subsequent nodes having smaller influenced expectations, i.e., $Ie(v_i, S) = \max_{v \in V_{\text{MIC}_{v_i}}} Ie(v, S)$. In other words, for any node v, it holds that $Ie(v, S) \geq Ie(\text{MIC}_v, S)$.

Algorithm 3 LocalSearch

Input: a graph $G(V_G, E_G)$, two parameters k and l, a size constraint γ, the influenced expectation $Ie(v, S)$ for $\forall v \in V_G$
Output: the most influenced community MIC with $|MIC| \geq \gamma$

1: $MIC \leftarrow \emptyset$; Queue $Q \leftarrow \emptyset$;
2: Compute the maximal D-core D of G;
3: Sort all nodes v of D's connected components $D_i \subseteq D$ with $|D_i| \geq \gamma$ in descending order of $Ie(v, S)$ and add them into Q;
4: **while** $Q \neq \emptyset$ **do**
5: Pop a vertex v from Q;
6: **if** $Ie(v, S) < Ie(MIC, S)$ **then Break**; //*Pruning strategy 1*
7: **if** v is pruned **then Continue**;
8: MIC$_v \leftarrow$ **SearchLocalMIC** (v, D);
9: **if** $Ie(MIC_v, S) > Ie(MIC, S)$ **then**
10: $MIC \leftarrow MIC_v$;
11: $D' \leftarrow$ Remove v from D and iteratively remove vertices that do not satisfy degree constraints of D-core;
12: **for** each node $v' \in V_D - V_{D'}$ **do** // *Pruning strategy 2(i)*
13: Mark v' as pruned.
14: **for** each connected component D'_i of D' **do**
15: **if** $|D'_i| < \gamma$ **then** // *Pruning strategy 2(ii)*
16: **for** each node $v' \in V_{D'_i}$ **do**
17: Mark v' as pruned.
18: **Return** MIC.

If $Ie(\text{MIC}_{v_i}, S) > Ie(v_{i+1}, S)$, then $Ie(\text{MIC}_{v_i}, S)$ will also be greater than $Ie(\text{MIC}_{v_j}, S)$ for all $j \geq i+1$. Consequently, node v_{i+1} and those following it can be safely pruned.

Pruning strategy 2. After computing MIC$_v$, node v is removed from D. This removal may result in some vertices' in-degree and out-degree failing to meet the degree constraints of D-core. To address this, we can iteratively remove these vertices to obtain a new D-core $D' \subset D$. Specifically, (i) Vertices in $V_D - V_{D'}$ can be pruned, as there will be no D-core containing these vertices, implying their local most influenced communities are empty. (ii) For each connected component D'_i of D', if $|D'_i| < \gamma$, all vertices of D'_i can be pruned, since we only consider D-core of size at least γ.

Pruning strategy 3. Evaluating D-core can be computationally expensive each time a node is added to MIC$_v$. We propose the following rules to reduce unnecessary calculations: (i) After adding a node u to MIC$_v$, if u's in-degree or out-degree in MIC$_v$ do not satisfy the degree constraints of D-core, we can skip the D-core examination and proceed to the next node. (ii) When peeling MIC$_v$ to check whether MIC$_v$ contains a D-core, if v is removed from MIC$_v$, the peeling process can be ended early. (iii) If MIC$_v$ contains fewer than γ nodes, we can skip the D-core examination.

The pseudo-code for the LocalSearch algorithm is provided in Algorithm 3. The algorithm begins by computing the maximal D-core D of the graph G, sorting all nodes v within the connected components $D_i \subseteq D$ with $|D_i| \geq \gamma$ by descending influenced expectations $Ie(v, S)$, and enqueueing them into an empty queue Q. Algorithm 3 then processes the nodes as follows. It dequeues a vertex v from Q and checks whether to compute MIC_v (Lines 6-7). If the computation is warranted, it calculates MIC_v using the function SearchLocalMIC and updates MIC (Lines 8-10). The algorithm then removes v from D, maintains D by iteratively removing vertices that do not meet degree constraints of D-core, and computes a new D-core D'. Using both D and D', Algorithm 3 prunes vertices whose local most influenced communities cannot contribute to the final result, following Pruning Strategy 2 (Lines 12-17). This process continues until all vertices in Q are examined, after which it returns MIC (Line 18).

Algorithm 4 SearchLocalMIC

Input: v, D
Output: MIC_v

1: $D_v \leftarrow$ the connected component of D containing v;
2: **for** each node $u \in V_{D_v}$ **do**
3: Mark u as unvisited;
4: $MIC_v \leftarrow \emptyset$; Queue $Q' \leftarrow \emptyset$; $G' \leftarrow \emptyset$;
5: Add v into Q;
6: **while** $Q \neq \emptyset$ **do**
7: Pop u from Q; Mark u as visited;
8: $G' \leftarrow G' \oplus u$;
9: **if** $d^{in}_{G'}(u) \geq k \wedge d^{out}_{G'}(u) \geq l \wedge |V_{G'}| \geq \gamma$ **then**
10: $D'_v \leftarrow$ Compute the connected D-core of G' that contains v and has a size of at least γ;
11: **if** $D'_v \neq \emptyset$ **then**
12: $MIC_v \leftarrow D'_v$; **Break**;
13: $N(u) \leftarrow$ all unvisited neighbors of u in D_v;
14: Sort all nodes w in $N(u)$ in descending order of $Ie(w, S)$;
15: **for** each node $w \in N(u)$ **do**
16: Add w into Q; Mark w as visited;
17: **Return** MIC_v.

Algorithm 4 shows the pseudo-code of function SearchLocalMIC. The function identifies the connected component D_v containing node v from the D-core D (Line 1) and marks all nodes in D_v as unvisited (Lines 2-3). It then initializes the D-core by adding node v to the queue Q (Line 5). Next, it dequeues a node u from Q, marks it as visited, and adds it to G' (Lines 7-8). If u meets the degree constraints and $V_{G'}$ has at least γ nodes, it computes the D-core D'_v in G' that contains v and has a size of at least γ (Lines 9-10). If D'_v is non-empty, it is returned as the local most influenced community of v (Lines 11-12). Otherwise, the function sorts all u's unvisited neighbors by descending influenced expectations and adds them to Q (Lines 13-16). The process continues until MIC_v is found or Q is empty.

Time Complexity. Following the complexity analysis of Algorithm 2, Algorithm 4 has a worst-case time complexity of $O(|L_v|(|V_G|+|E_G|))$. With ω nodes having lower influenced expectation than the MIC, Algorithm 3 computes the local most influenced community for $(|V_G|-\omega)$ nodes. Thus, the time complexity of Algorithm 3 is $O(|L_v|(|V_G|-\omega)(|V_G|+|E_G|))$.

VII. Experiments

In this section, we empirically evaluate our proposed algorithms. All implementations are done in C++ and executed on a Linux server with 3.70GHz CPU and 128GB of RAM.

Our experiments utilize eight real-world directed networks, detailed in Table I. The *WikiLink* is from Konect [66], while the others are available on SNAP [67]. For each network, we apply the *weighted-cascade* model [29] to compute the influence weight $w(u, v)$ for each edge, i.e., $w(u, v) = \frac{1}{|d^{in}_G(v)|}$.

Following [68], we generate two types of seed node sets: *skewed* and *random*. The skewed seeds consist of the most influential nodes representing popular or influential users within the network. In contrast, the random seeds are sampled randomly from the network. Since the results from these two

TABLE I
DATASET STATISTICS

Dataset	n	m	d_{avg}	k_{max}	l_{max}
EmailCore (EC)	1,005	25,571	49.6	27	26
WikiVote (WV)	7,115	103,689	14.57	19	15
Epinions (EP)	75,879	508,837	6.7	40	42
Slashdot (SD)	82,168	948,464	11.54	54	9
EmailEuall (EE)	265,214	420,045	1.58	28	28
Pokec (PO)	1,632,803	30,622,564	18.75	32	31
LiveJournal (LJ)	4,847,571	68,993,773	14.13	252	253
WikiLink (WL)	13,593,032	437,217,424	64.33	1086	1086

types of seed nodes are similar, we only present the results from the skewed seed set for brevity. Detailed results from the random seed set and ***all source codes*** can be found in [25].

A. Performance of the Predictive Models

In our first set of experiments, we evaluate the performance of the predictive models used in L-InfExp algorithm. We assess three models: Random Forest (RF) [63], Decision Tree (DT) [65], and k-Nearest Neighbor (KNN) [64]. Specifically, the Decision Tree predicts outcomes by recursively splitting data into branches based on feature values; the Random Forest constructs multiple decision trees and aggregates their predictions to improve prediction accuracy and reduce overfitting; the k-Nearest Neighbor predicts the value of a data point based on the values of its k nearest neighbors.

We employ Random Forest in L-InfExp due to its superior predictive performance, as demonstrated in Exp-1. We set the default sample size to 20 and use the following hyperparameters for Random Forest: *NE*=100, *MD*=None, *MSL*=1, and *MSS*=2. Specifically, *NE* refers to the number of decision trees; *MD* indicates the maximum depth of each decision tree; *MSL* and *MSS* represent the minimum number of samples required to split a leaf node and an internal node, respectively.

Exp-1: Performance of predictive models. In this experiment, we compare RF, DT, and KNN. We generate 20 seed node sets and compute the influenced expectations for each set. Of these, 15 sets are used for training and the remaining 5 for testing. For each model, we report both *training time* (T-time) and *prediction time* (P-time), as well as *predictive accuracy* using four metrics: *Mean Squared Error* (MSE), *Root Mean Squared Error* (RMSE), *Mean Absolute Error* (MAE), and *R-squared*. For MSE, RMSE, and MAE, lower values indicate better performance. For R-squared, higher values are preferred. The results, summarized in Table II, show that RF delivers the best predictive accuracy among the three models, with the smallest disparity between predicted and actual influenced expectations. Therefore, we employ RF in the L-InfExp algorithm due to its superior predictive accuracy.

Exp-2: Effect of hyperparameters on RF model. We study the impact of various hyperparameters combinations on RF performance as the number of seed nodes $|S|$ varies. The four hyperparameters tested are MD in {None, 10, 20}, MSL in {1, 2, 4}, MSS in {2, 5, 10}, and NE in {50, 100, 200}. A grid search is used to evaluate performance by testing all possible combinations of these hyperparameter and employing cross-validation to identify the best performing set. The optimal hyperparameter combinations for different $|S|$ values across different datasets are presented in Table III. It can be observed that, smaller MSL and MSS values generally improve performance by producing more complex decision trees, which fit the data more accurately, while MD and NE have a much smaller impact on the model's accuracy.

Exp-3: Sensitivity analysis of RF model. We conduct a sensitivity analysis of RF on *Epinions* by varying four key hyperparameters: MD, MSL, MSS, and NE. We randomly select 50% of the samples for training and the remaining 50% for testing. Results in Table IV show that increasing MSL and MSS reduces training time, as larger values require fewer split, thereby reducing tree depth and model complexity, and thereby lowering the computational cost of finding optimal splits. However, prediction time remains relatively stable, since the number of trees and their traversal paths do not change significantly. Higher MSL and MSS lower prediction performance due to model simplification. Increasing NE raises both training and prediction times, while MD and NE have little effect on accuracy. Lower MSL and MSS improve performance. Based on these findings, the default hyperparameters are set as: MD=None, MSL=1, MSS=2, and NE=100.

Exp-4: Fit analysis of RF model. We conduct a fit analysis experiment of the RF model on the *Epinions* by varying MD, MSL, MSS, NE, and sample size. We use 50% of the samples for training and the remaining 50% for testing, with MSE as the evaluation metric. Results are plotted in Figure 7. We observe that increasing MSL and MSS leads to higher training and testing errors, indicating that reduced model complexity decreases fit quality. Additionally, increasing MD and NE has little effect on training or testing errors. Furthermore, a larger sample size improves model stability and generalization ability, as evidenced by lower testing errors.

B. Efficiency Evaluation

In this set of experiments, we evaluate the efficiency of our proposed algorithms. Due to space constraint and similar results, we present results for the *EmailEuall* and *Wiki-Link* in this paper. Results for other datasets can be found in [25].

Algorithms. To our best knowledge, this is the first work studying the Most Influenced Community search in large social graphs. Hence, for comparative methods, we extend two widely-used existing algorithms: Monte-Carlo simulation (MCS) [29], designed for influence maximization, and local search [40], developed for identifying the top-r influential communities, to address our problem. Therefore, we compare six methods listed as follows:

For **Phase I**-Influenced expectation computation:
- **S-InfExp**: our proposed sampling-based algorithm.
- **L-InfExp**: our proposed learning-based algorithm.
- **M-InfExp**: an extended MCS-based algorithm [29].

For **Phase II**-MIC search:
- **GlobalSearch**: our proposed global search algorithm.
- **LocalSearch**: our proposed local search algorithm.
- **NeiborSearch**: a method extended from the local search algorithm proposed in [40].

Parameters. We test parameters including D-core integers k and l, seed set size $|S|$, community size threshold γ, graph size

TABLE II
PERFORMANCE OF PREDICTING MODELS

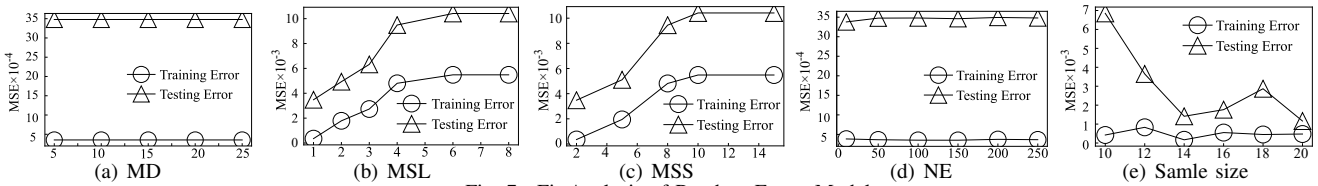
Model	Pokec						Livejournal						WikiLink					
	T-time	**P-time**	**MSE**	**RMSE**	**MAE**	**R-Squared**	**T-time**	**P-time**	**MSE**	**RMSE**	**MAE**	**R-Squared**	**T-time**	**P-time**	**MSE**	**RMSE**	**MAE**	**R-Squared**
RF	701.65s	0.0964s	**6.29E-05**	**0.0079**	**0.0031**	**0.9909**	2241.8s	0.5089s	**5.26E-05**	**0.0072**	**0.0023**	**0.9891**	6309.8s	1.4454s	**6.47E-05**	**0.0080**	**0.0018**	**0.9919**
DT	21.906s	**0.0024s**	0.0005	0.0229	0.0093	0.9238	70.219s	**0.0119s**	0.0003	0.0198	0.0064	0.9186	200.98s	**0.0393s**	0.0004	0.0209	0.0044	0.9452
KNN	**0.0433s**	0.0535s	0.0002	0.0139	0.0052	0.9720	**0.1142s**	0.1664s	0.0001	0.0122	0.0036	0.9692	**0.1715s**	0.4785s	0.0002	0.0150	0.0027	0.9716

TABLE III
EFFECT OF PARAMETERS OF RANDOM FOREST MODEL

$\|S\|$	EmailCore					$\|S\|$	Pokec					$\|S\|$	Livejournal					$\|S\|$	WikiLink				
	MD	**MSL**	**MSS**	**NE**	**MSE**		**MD**	**MSL**	**MSS**	**NE**	**MSE**		**MD**	**MSL**	**MSS**	**NE**	**MSE**		**MD**	**MSL**	**MSS**	**NE**	**MSE**
10	None	1	5	50	0.0154	**100**	None	4	5	100	0.0028	**100**	10	1	10	50	0.0021	**100**	10	1	5	50	0.0021
20	None	1	10	50	0.0028	**200**	10	1	2	50	0.0029	**200**	10	1	2	100	0.0021	**200**	None	1	5	50	0.0029
30	None	1	5	50	0.0188	**300**	10	1	5	100	0.0051	**300**	10	1	10	50	0.0035	**300**	None	1	10	50	0.0035
40	20	2	5	50	0.0207	**400**	10	4	5	50	0.0060	**400**	10	1	5	50	0.0042	**400**	None	1	2	100	0.0042
50	20	1	5	50	0.0249	**500**	20	1	10	50	0.0067	**500**	None	1	2	100	0.0048	**500**	None	1	5	100	0.0048

TABLE IV
SENSITIVITY ANALYSIS OF RANDOM FOREST MODEL

MD	**T-time**	**P-time**	**MSE**	**R-Squard**	**MSL**	**T-time**	**P-time**	**MSE**	**R-Squard**	**MSS**	**T-time**	**P-time**	**MSE**	**R-Squard**	**NE**	**T-time**	**P-time**	**MSE**	**R-Squard**
None	4.5851	0.009	0.0034	0.7594	**1**	4.5986	0.0091	0.0034	0.7594	**2**	4.7158	0.0091	0.0034	0.7594	**50**	2.3972	0.0048	0.0034	0.7618
5	4.5882	0.009	0.0034	0.7594	**2**	2.2397	0.01	0.0049	0.6795	**5**	2.7278	0.009	0.0051	0.6828	**100**	4.7405	0.0091	0.0034	0.7594
10	4.5853	0.009	0.0035	0.7594	**4**	0.6106	0.009	0.0094	0.3915	**8**	0.7821	0.009	0.0094	0.3974	**150**	7.0352	0.0133	0.0033	0.7465
20	4.5896	0.009	0.0034	0.7594	**6**	0.449	0.009	0.0104	0.3307	**10**	0.4473	0.009	0.0104	0.3307	**200**	9.2079	0.0175	0.0034	0.7478



(a) MD (b) MSL (c) MSS (d) NE (e) Samle size

Fig. 7. Fit Analysis of Random Forest Model

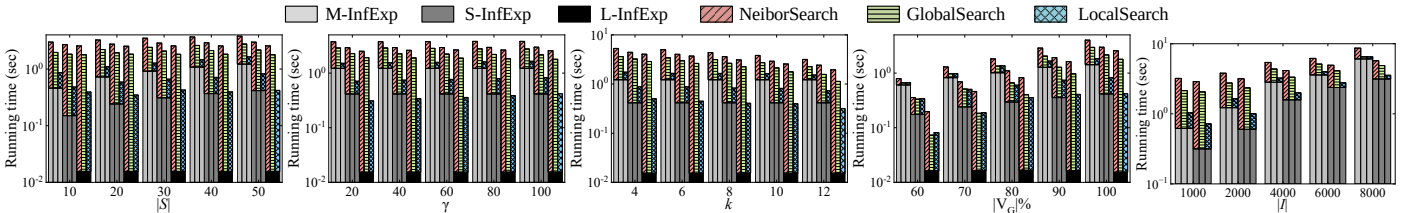
$|V_G|$, and influence propagation instances $|\mathcal{I}|$. Each experiment varies one parameter while keeping others at default values. For small datasets such as *EmailCore*, *WikiVote*, *Epinions*, *Slashdot*, and *EmailEuall*, we set $|S| = 50$; for large datasets like *Pokec*, *LiveJournal*, and *WikiLink*, $|S| = 500$. Default values for $|\mathcal{I}|$ and γ are 2000 and 100, respectively. We terminate an algorithm if it cannot finish in 24 hours and use **OOT** (i.e., out of time) to represent the result.

Exp-5: Effect of $|S|$. We analyze how the number of seed nodes $|S|$ affects running time. Results are shown in Figures 8(a) and 9(a). We observe that the running times of M-InfExp and S-InfExp increase with $|S|$, as they need to generate more influence propagation instances, resulting in higher running time. In addition, it can be seen that S-InfExp runs faster than M-InfExp because S-InfExp skips certain nodes when sampling influence propagation instances, thereby improving sampling efficiency. L-InfExp, on the other hand, maintains a relatively stable and lower running time. This is because, for online influenced expectation prediction, S-InfExp encodes S into a multi-hot vector $\boldsymbol{x}_S \in \{0,1\}^n$, so its performance is less sensitive to the changes in $|S|$. Moreover, the running times of NeiborSearch, GlobalSearch, and LocalSearch remain relatively stable with $|S|$ since this parameter mainly influences the computation of influenced expectations. Notably, LocalSearch is significantly faster than GlobalSearch and NeiborSearch across all settings, reflecting the high efficiency of LocalSearch algorithm.

Exp-6: Effect of γ. We investigate the impact of the size constraint γ on algorithm performance. Figures 8(b) and 9(b) show the results. We can observe that increasing γ has little effect on running time across all algorithms for influenced expectation computation. This is because γ only affects the size of the returned community, not the computation of influenced expectations. Moreover, the running time of GlobalSearch decreases as γ increases. This is because a larger γ leads to newly generated connected components that are less likely to satisfy the size condition, thereby reducing the number of communities that need to be examined. However, both NeiborSearch and LocalSearch show a slight increase in running time with larger γ. This is due to the fact that, more vertices need to be inserted to search for the local most influenced community that meets the larger size condition, resulting in increased time required to compute the D-core and traverse all connected components.

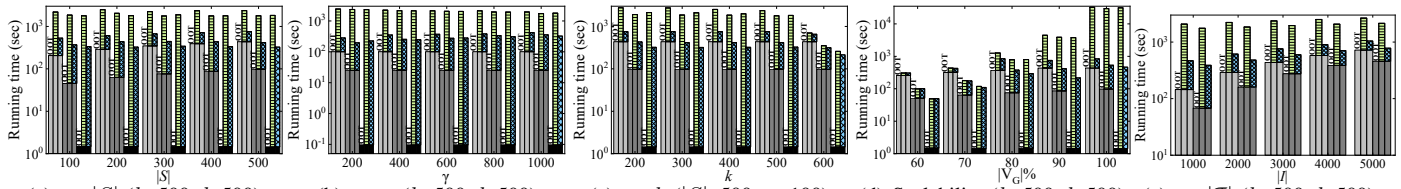
Exp-7: Effect of k and l. We evaluate the effects of parameters k and l on algorithm performance. Figures 8(c) and 9(c) present the results for k. The results for l, which are similar to the results for k, are reported in our technical report [25] because of space constraints. As k (or l) increases, the running time for NeiborSearch, GlobalSearch, and LocalSearch decreases because a larger k results in a smaller D-core, reducing the number of nodes that need to be processed. The running times of M-InfExp, S-InfExp, and L-InfExp remain unchanged as k and l only impact the community search, not the computation of influenced expectations.

Exp-8: Scalability. We assess the scalability of our algorithms by varying the graph size. To this end, we randomly select a certain number of nodes from the graph to form the induced subgraph and test the running time of all algorithms. Figures 8(d) and 9(d) illustrate that the running times of all algorithms, except for L-InfExp, increase as the graph size grows. This is because NeiborSearch, GlobalSearch, and LocalSearch require more time to identify communities in larger graphs. For M-InfExp and S-InfExp, the increased running time is mainly due to the larger size of influence propagation instances. The running time of L-InfExp remains



(a) vs. $|S|$ (k=10, l=10) (b) vs. γ (k=10, l=10) (c) vs. k ($|S|$=50, γ=100) (d) Scalability (k=10, l=10) (e) vs. $|\mathcal{I}|$ (k=10, l=10)

Fig. 8. Running time on EmailEuall



(a) vs. $|S|$ (k=500, l=500) (b) vs. γ (k=500, l=500) (c) vs. k ($|S|$=500, γ=100) (d) Scalability (k=500, l=500) (e) vs. $|\mathcal{I}|$ (k=500, l=500)

Fig. 9. Running time on WikiLink

TABLE V
INFLUENCED EXPECTATIONS COMPARISON

Dataset	Avg Function															Min Function					
	MIC									Original Graph			Maximal D-core			MIC					
	M+G	M+L	M+N	S+G	S+L	S+N	L+G	L+L	L+N	M-INF	S-INF	L-INF	M-INF	S-INF	L-INF	M+G	M+L	S+G	S+L	L+G	L+L
EC ($k=10$, $l=10$, $\gamma=20$)	0.9748	0.9212	0.9718	**0.9749**	0.9312	0.9742	0.9348	0.8753	0.9348	0.4409	0.4122	0.4108	0.4588	0.4317	0.4300	0.4963	0.4963	0.4970	0.4970	0.5220	0.5220
WV ($k=5$, $l=5$, $\gamma=20$)	0.5738	0.4526	0.5738	0.5628	0.6019	0.5370	0.6044	**0.6332**	0.4566	0.0833	0.0804	0.0804	0.2762	0.2699	0.2702	0.2727	0.2727	0.2720	0.2720	0.2700	0.2700
EP ($k=20$, $l=20$, $\gamma=20$)	**0.2516**	0.2438	0.2412	0.2312	0.2175	0.2176	0.2246	0.2122	0.2173	0.1572	0.1410	0.1387	0.2152	0.1882	0.1836	0.1997	0.1997	0.1760	0.1760	0.1720	0.1720
SD ($k=40$, $l=5$, $\gamma=20$)	**0.2808**	0.2770	0.2721	0.2639	0.2640	0.2639	0.2704	0.2763	0.2621	0.2100	0.1813	0.1829	0.2621	0.2445	0.2472	0.2070	0.2070	0.1890	0.1890	0.1920	0.1920
EE ($k=10$, $l=10$, $\gamma=20$)	**0.7564**	0.6781	0.7534	0.7344	0.7239	0.7344	0.7480	0.6641	0.7460	0.0651	0.0618	0.0613	0.1822	0.1561	0.1544	0.1690	0.1690	0.1350	0.1350	0.1520	0.1520
PO ($k=23$, $l=23$, $\gamma=100$)	**0.2422**	0.2393	0.2322	0.2364	0.2186	0.2364	0.2353	0.2187	0.2353	0.0954	0.0720	0.0713	0.1045	0.0871	0.0869	0.1723	0.1723	0.1590	0.1590	0.1600	0.1600
LJ ($k=75$, $l=75$, $\gamma=100$)	**0.2547**	0.2534	0.2530	0.2465	0.2460	0.2465	0.2412	0.2407	0.2412	0.0487	0.0375	0.0364	0.0188	0.0174	0.0169	0.2017	0.2017	0.1920	0.1920	0.1910	0.1910
WL ($k=500$, $l=500$, $\gamma=100$)	**0.0508**	0.0507	0.0501	0.0399	0.0397	0.0399	0.0389	0.0388	0.0389	0.0350	0.0299	0.0293	0.0104	0.0089	0.0088	0.0092	0.0092	0.0050	0.0050	0.0060	0.0060

* **M**/**S**/**L**+**G**/**L**/**N**: **M**-InfExp/**S**-InfExp/**L**-InfExp+**G**lobalSearch/**L**ocalSearch/**N**eiborSearch.

stable, as the efficiency of this learning based algorithm is not sensitive to the graph size.

Exp-9: Effect of $|\mathcal{I}|$. We evaluate the impact of the number of influence propagation instances $|\mathcal{I}|$ on algorithm efficiency. The results are shown in Figures 8(e) and 9(e). We observe that the running times of M-InfExp and S-InfExp increase with $|\mathcal{I}|$, reflecting their time complexity. In addition, S-InfExp runs faster than M-InfExp across all settings. In contrast, NeiborSearch, GlobalSearch, and LocalSearch show stable running times as $|\mathcal{I}|$ varies, since $|\mathcal{I}|$ mainly affects the computation of influenced expectations, which is independent of the MIC search.

C. Effectiveness Evaluation

In this set of experiments, we evaluate the quality of returned communities by using different aggregation functions.

Exp-10: Quality of MIC. In this experiments, we evaluate the influenced expectation of the most influenced community (MIC) using two aggregation functions: *avg* and *min*. We extend our GlobalSearch and LocalSearch to find the community defined under the *min* aggregation function. For comparisons, we also report the influenced expectations for the entire graph and the maximal D-core. We set all parameters to their default values. The results are summarized in Table V. From Table V, we make the following observations. **(1)** The MIC identified by the GlobalSearch algorithm has a higher influenced expectation compared to the MIC found by LocalSearch and NeiborSearch. **(2)** The influenced expectation of the MIC computed by the L-InfExp algorithm is very close to that of the MIC calculated by S-InfExp, indicating the prediction accuracy of the L-InfExp algorithm. **(3)** The expected influence of the MIC computed by the M-InfExp algorithm is slightly higher than that of S-InfExp, as M-InfExp samples the entire graph to generate influence propagation instances. **(4)** The influenced expectation of the MIC is significantly greater than that of the whole graph and the maximal D-core, suggesting that nodes in the MIC are more likely to be influenced by the seed nodes. **(5)** The influenced expectation of the MIC found by the NeiborSearch method is lower than that found using our methods, demonstrating the superiority of our proposed algorithms. **(6)** The influenced expectation of the MIC under function *min* is lower than that of the MIC under function *avg*, consistent with our expectation. Overall, the community identified by function *avg* and our proposed algorithms demonstrate the best quality.

VIII. CONCLUSION

In this paper, we introduce the Most Influenced Community Search (MICS) problem in social networks for the first time. We propose two algorithms, S-InfExp and L-InfExp, which compute the influenced expectation of each node using sampling and learning techniques, respectively. To identify the most influenced community based on these expectations, we further develop GlobalSearch and LocalSearch. Specifically, GlobalSearch follows a top-down approach, while LocalSearch adopts a bottom-up strategy. Extensive experiments on eight large real-world networks show that our algorithms are effective, efficient, and scalable. This research marks our initial step in the study of influenced community search. Moving forward, we aim to explore the dynamics of influenced community search within evolving social networks.

REFERENCES

[1] F. Zhou, X. Xu, G. Trajcevski, and K. Zhang, "A survey of information cascade analysis: Models, predictions, and recent advances," *ACM Computing Surveys (CSUR)*, vol. 54, no. 2, pp. 1–36, 2021.

[2] Y. Li, J. Fan, Y. Wang, and K.-L. Tan, "Influence maximization on social graphs: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 10, pp. 1852–1872, 2018.

[3] R. Bian, Y. S. Koh, G. Dobbie, and A. Divoli, "Identifying top-k nodes in social networks: a survey," *ACM Computing Surveys (CSUR)*, vol. 52, no. 1, pp. 1–33, 2019.

[4] Y. Yang and J. Pei, "Influence analysis in evolving networks: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 33, no. 3, pp. 1045–1063, 2019.

[5] H. Li, C. Xia, T. Wang, S. Wen, C. Chen, and Y. Xiang, "Capturing dynamics of information diffusion in sns: A survey of methodology and techniques," *ACM Computing Surveys (CSUR)*, vol. 55, no. 1, pp. 1–51, 2021.

[6] F. Zhou, X. Xu, K. Zhang, G. Trajcevski, and T. Zhong, "Variational information diffusion for probabilistic cascades prediction," in *IEEE INFOCOM 2020-IEEE conference on computer communications*. IEEE, 2020, pp. 1618–1627.

[7] T. R. Zaman, R. Herbrich, J. Van Gael, and D. Stern, "Predicting information spreading in twitter," in *Workshop on computational social science and the wisdom of crowds, nips*, vol. 104, no. 45. Citeseer, 2010, pp. 17 599–601.

[8] B. Zhang, Z. Qian, and S. Lu, "Structure pattern analysis and cascade prediction in social networks," in *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD*. Springer, 2016, pp. 524–539.

[9] S. Huang, Z. Bao, J. S. Culpepper, and B. Zhang, "Finding temporal influential users over evolving social networks," in *2019 IEEE 35th international conference on data engineering (ICDE)*. IEEE, 2019, pp. 398–409.

[10] J. Zhao, S. Shang, P. Wang, J. C. Lui, and X. Zhang, "Tracking influential nodes in time-decaying dynamic interaction networks," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 2019, pp. 1106–1117.

[11] Y. Zhu, J. Tang, and X. Tang, "Pricing influential nodes in online social networks," *Proceedings of the VLDB Endowment*, vol. 13, no. 10, pp. 1614–1627, 2020.

[12] Y. Wang, G. Cong, G. Song, and K. Xie, "Community-based greedy algorithm for mining top-k influential nodes in mobile social networks," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2010, pp. 1039–1048.

[13] F. Riquelme and P. González-Cantergiani, "Measuring user influence on twitter: A survey," *Information processing & management*, vol. 52, no. 5, pp. 949–975, 2016.

[14] N. Hafiene, W. Karoui, and L. B. Romdhane, "Influential nodes detection in dynamic social networks: A survey," *Expert Systems with Applications*, vol. 159, p. 113642, 2020.

[15] Q. Guo, C. Feng, F. Zhang, and S. Wang, "Efficient algorithm for budgeted adaptive influence maximization: An incremental rr-set update approach," *Proceedings of the ACM on Management of Data*, vol. 1, no. 3, pp. 1–26, 2023.

[16] C. Borgs, M. Brautbar, J. Chayes, and B. Lucier, "Maximizing social influence in nearly optimal time," in *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*. SIAM, 2014, pp. 946–957.

[17] Y. Tang, X. Xiao, and Y. Shi, "Influence maximization: Near-optimal time complexity meets practical efficiency," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, 2014, pp. 75–86.

[18] Y. Tang, Y. Shi, and X. Xiao, "Influence maximization in near-linear time: A martingale approach," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 2015, pp. 1539–1554.

[19] J. Tang, X. Tang, X. Xiao, and J. Yuan, "Online processing algorithms for influence maximization," in *Proceedings of the 2018 ACM SIGMOD International Conference on Management of Data*, 2018, pp. 991–1005.

[20] Q. Guo, S. Wang, Z. Wei, and M. Chen, "Influence maximization revisited: Efficient reverse reachable set generation with bound tightened," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 2167–2181.

[21] SBDC, "Why target marketing is important and how to do it." 2024. [Online]. Available: https://georgiasbdc.org/why-target-marketing-is-important-and-how-to-do-it/

[22] X. Yao, X. Han, Y. Gu, C. Gu, and H. Huang, "Influence spread in location-based social network: An efficient algorithm of epidemic controlling," *IEEE Transactions on Computational Social Systems*, vol. 9, no. 4, pp. 1132–1143, 2021.

[23] A. Goyal, F. Bonchi, and L. V. Lakshmanan, "Learning influence probabilities in social networks," in *Proceedings of the third ACM international conference on Web search and data mining*, 2010, pp. 241–250.

[24] D. J. Watts, J. Peretti, and M. Frumin, *Viral marketing for the real world*. Harvard Business School Pub. Boston, 2007.

[25] X. Chang, Q. Liu, Y. Gao, B. Zheng, Y. Cai, and Q. Li, "The most influenced community search on social networks," 2024. [Online]. Available: https://github.com/ZJU-DAILY/MICS

[26] C. Giatsidis, D. M. Thilikos, and M. Vazirgiannis, "D-cores: measuring collaboration of directed graphs based on degeneracy," *Knowledge and information systems*, vol. 35, no. 2, pp. 311–343, 2013.

[27] X. Liao, Q. Liu, J. Jiang, X. Huang, J. Xu, and B. Choi, "Distributed d-core decomposition over large directed graphs," *Proceedings of the VLDB Endowment*, vol. 15, no. 8, pp. 1546–1558, 2022.

[28] Y. Fang, Z. Wang, R. Cheng, H. Wang, and J. Hu, "Effective and efficient community search over large directed graphs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 11, pp. 2093–2107, 2018.

[29] D. Kempe, J. Kleinberg, and É. Tardos, "Maximizing the spread of influence through a social network," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2003, pp. 137–146.

[30] H. B. S. O. B. Insights, "Why identifying your target audience is important to your marketing strategy." 2024. [Online]. Available: https://online.hbs.edu/blog/post/target-audience-in-marketing

[31] J. Brown, A. J. Broderick, and N. Lee, "Word of mouth communication within online communities: Conceptualizing the online social network," *Journal of interactive marketing*, vol. 21, no. 3, pp. 2–20, 2007.

[32] R. M. Christley, G. Pinchbeck, R. G. Bowers, D. Clancy, N. P. French, R. Bennett, and J. Turner, "Infection in social networks: using network analysis to identify high-risk individuals," *American journal of epidemiology*, vol. 162, no. 10, pp. 1024–1031, 2005.

[33] K. Bringmann and K. Panagiotou, "Efficient sampling methods for discrete distributions," *Algorithmica*, vol. 79, pp. 484–508, 2017.

[34] S. Chen, R. Wei, D. Popova, and A. Thomo, "Efficient computation of importance based communities in web-scale networks using a single machine," in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, 2016, pp. 1553–1562.

[35] R.-H. Li, L. Qin, J. X. Yu, and R. Mao, "Finding influential communities in massive networks," *The VLDB Journal*, vol. 26, pp. 751–776, 2017.

[36] W. Luo, X. Zhou, J. Yang, P. Peng, G. Xiao, and Y. Gao, "Efficient approaches to top-r influential community search," *IEEE Internet of Things Journal*, vol. 8, no. 16, pp. 12 650–12 657, 2020.

[37] Y. Wu, J. Zhao, R. Sun, C. Chen, and X. Wang, "Efficient personalized influential community search in large networks," *Data Science and Engineering*, vol. 6, no. 3, pp. 310–322, 2021.

[38] R.-H. Li, L. Qin, J. X. Yu, and R. Mao, "Influential community search in large networks," *Proceedings of the VLDB Endowment*, vol. 8, no. 5, pp. 509–520, 2015.

[39] R.-H. Li, L. Qin, F. Ye, J. X. Yu, X. Xiao, N. Xiao, and Z. Zheng, "Skyline community search in multi-valued networks," in *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 457–472.

[40] Y. Peng, S. Bian, R. Li, S. Wang, and J. X. Yu, "Finding top-r influential communities under aggregation functions," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2022, pp. 1941–1954.

[41] Y. Zhou, Y. Fang, W. Luo, and Y. Ye, "Influential community search over large heterogeneous information networks," *Proceedings of the VLDB Endowment*, vol. 16, no. 8, pp. 2047–2060, 2023.

[42] F. Bi, L. Chang, X. Lin, and W. Zhang, "An optimal and progressive approach to online search of top-k influential communities," *Proceedings of the VLDB Endowment*, vol. 11, no. 9, 2018.

[43] Y. Fang, X. Huang, L. Qin, Y. Zhang, W. Zhang, R. Cheng, and X. Lin, "A survey of community search over big graphs," *The VLDB Journal*, vol. 29, pp. 353–392, 2020.

[44] Q. Liu, M. Zhao, X. Huang, J. Xu, and Y. Gao, "Truss-based community search over large directed graphs," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 2183–2197.

[45] X. Huang and L. V. Lakshmanan, "Attribute-driven community search," *Proceedings of the VLDB Endowment*, vol. 10, no. 9, pp. 949–960, 2017.

[46] L. Chen, C. Liu, R. Zhou, J. Li, X. Yang, and B. Wang, "Maximum co-located community search in large scale social networks," *Proceedings of the VLDB Endowment*, vol. 11, no. 10, pp. 1233–1246, 2018.

[47] Q. Liu, Y. Zhu, M. Zhao, X. Huang, J. Xu, and Y. Gao, "Vac: vertex-centric attributed community search," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 2020, pp. 937–948.

[48] Y. Fang, Y. Yang, W. Zhang, X. Lin, and X. Cao, "Effective and efficient community search over large heterogeneous information networks," *Proceedings of the VLDB Endowment*, vol. 13, no. 6, pp. 854–867, 2020.

[49] Y. Jiang, Y. Fang, C. Ma, X. Cao, and C. Li, "Effective community search over large star-schema heterogeneous information networks," *Proceedings of the VLDB Endowment*, vol. 15, no. 11, pp. 2307–2320, 2022.

[50] J. Wang, K. Wang, X. Lin, W. Zhang, and Y. Zhang, "Efficient unsupervised community search with pre-trained graph transformer," *Proceedings of the VLDB Endowment*, vol. 17, no. 9, pp. 2227–2240, 2024.

[51] Y. Jiang, Y. Rong, H. Cheng, X. Huang, K. Zhao, and J. Huang, "Query driven-graph neural networks for community search: from non-attributed, attributed, to interactive attributed," *Proceedings of the VLDB Endowment*, vol. 15, no. 6, pp. 1243–1255, 2022.

[52] M. Girvan and M. E. Newman, "Community structure in social and biological networks," *Proceedings of the national academy of sciences*, vol. 99, no. 12, pp. 7821–7826, 2002.

[53] J. Li, H. Zhang, Z. Han, Y. Rong, H. Cheng, and J. Huang, "Adversarial attack on community detection by hiding individuals," in *Proceedings of The Web Conference 2020*, 2020, pp. 917–927.

[54] X. Jian, Y. Wang, and L. Chen, "Effective and efficient relational community detection and search in large dynamic heterogeneous information networks," *Proceedings of the VLDB Endowment*, vol. 13, no. 10, pp. 1723–1736, 2020.

[55] Z. Xu, Y. Ke, Y. Wang, H. Cheng, and J. Cheng, "A model-based approach to attributed graph clustering," in *Proceedings of the 2012 ACM SIGMOD international conference on management of data*, 2012, pp. 505–516.

[56] A. Conte, T. De Matteis, D. De Sensi, R. Grossi, A. Marino, and L. Versari, "D2k: scalable community detection in massive networks via small-diameter k-plexes," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 1272–1281.

[57] N. Veldt, D. F. Gleich, and A. Wirth, "A correlation clustering framework for community detection," in *Proceedings of the 2018 World Wide Web Conference*, 2018, pp. 439–448.

[58] X. Chang, X. Ke, L. Chen, C. Ge, Z. Wei, and Y. Gao, "Host profit maximization: Leveraging performance incentives and user flexibility," *Proceedings of the VLDB Endowment*, vol. 17, no. 1, pp. 51–64, 2023.

[59] Z. Liang, Y. Yang, X. Ke, X. Xiao, and Y. Gao, "A benchmark study of deep-rl methods for maximum coverage problems over graphs," *Proceedings of the VLDB Endowment*, vol. 17, no. 11, pp. 3666–3679, 2024.

[60] W. Chen, C. Wang, and Y. Wang, "Scalable influence maximization for prevalent viral marketing in large-scale social networks," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2010, pp. 1029–1038.

[61] O. Amini, D. Peleg, S. Pérennes, I. Sau, and S. Saurabh, "On the approximability of some degree-constrained subgraph problems," *Discrete Applied Mathematics*, vol. 160, no. 12, pp. 1661–1679, 2012.

[62] M. Mitzenmacher and E. Upfal, *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.

[63] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[64] P. Cunningham and S. J. Delany, "K-nearest neighbour classifiers-a tutorial," *ACM computing surveys (CSUR)*, vol. 54, no. 6, pp. 1–25, 2021.

[65] V. G. Costa and C. E. Pedreira, "Recent advances in decision trees: An updated survey," *Artificial Intelligence Review*, vol. 56, no. 5, pp. 4765–4800, 2023.

[66] J. Kunegis, "Konect: the koblenz network collection," in *Proceedings of the 22nd international conference on world wide web*, 2013, pp. 1343–1350.

[67] J. Leskovec and A. Krevl, "Snap datasets: Stanford large network dataset collection," 2014, http://snap.stanford.edu/data.

[68] M. Simpson, F. Hashemi, and L. V. Lakshmanan, "Misinformation mitigation under differential propagation rates and temporal penalties," *Proceedings of the VLDB Endowment*, vol. 15, no. 10, pp. 2216–2229, 2022.

APPENDIX

A. Aggregation Functions Extensions

In this section, we extend our problem and techniques to other aggregation functions, such as *max*, *min*, and *sum*. Specifically, the influenced expectation of a MIC is defined as the minimum, maximum, or sum of the influenced expectations of the nodes in that community. **(1)** For the max function in disease prevention, a "super-spreader" with the maximum influenced expectation can significantly increase the community's overall infection risk, as one highly influenced individual can impact the entire community. **(2)** For the min function, in a marketing campaign, the success of promoting a product relies on the purchasing desire of every community member, and the minimum influenced expectation ensures that all members of the community are engaged. **(3)** For the sum function, the total purchase probability of all community members in a marketing campaign reflects the overall demand for a product, indicating the potential market capacity.

Under the max and sum functions, the MIC straightforwardly corresponds to the maximal D-core in the initial graph G, provided the size of maximal D-core is larger than γ. Therefore, we focus on the minimum function. We begin with a modified definition of the influenced expectation of a subgraph H of G under the min function.

Definition 6. (Influenced expectation of H under min function). *Given a subgraph $H(V_H, E_H) \subseteq G$, a node set $S \subset V$, and an influence propagation model M, we denote the influenced expectation of H as $Ie(H, S) = \min_{v \in H_v} Ie(v, S)$, which represents the probability that H is influenced by S under propagation model M.*

It can be verified that the MIC defined in Definition 4 is also suitable for the above definition. Then, we extend our GlobalSearch and LocalSearch to solve the MICS problem under the min function based on the following Lemmas.

Lemma 1. *For any community C, if we delete the node with the smallest influenced expectation in C and the resulting subgraph still contains a community C', then the influenced expectation of C' is no smaller than that of C.*

Lemma 2. *Given an empty subgraph G', we add nodes from the maximal D-core of G to G' in descending order of their influenced expectations until G' contains a MIC C'. The influenced expectation of C' is then the maximal.*

The proofs of the above two Lemmas are straightforward, thus, we omit them here. Based on Lemma 1, our GlobalSearch algorithm can directly extend to solve the MICS problem under the min function. Then, we extend our LocalSearch algorithm based on Lemma 2 and propose the LocalSearch-min algorithm. The main ideas of this algorithm are as follows. We construct a subgraph G' by gradually adding nodes in descending order of their influenced expectations from the maximal D-core in original graph G, until a maximal connected subgraph is formed. This subgraph should be larger than γ and each node in it must satisfy the (k, l) constraint. Then, this connected subgraph is recognized as the MIC with the maximum influenced expectation. The pseudo-code of the LocalSearch-min algorithm is presented in Algorithm 5. Since it is easy to follow, we omit the detailed explanation here.

Algorithm 5 LocalSearch-min

Input: a graph $G(V_G, E_G)$, two parameters k and l, a size constraint γ, the influenced expectation $Ie(v, S)$ for $\forall v \in V_G$
Output: the most influenced community MIC with $MIC \geq \gamma$

```
Compute D-core D of G
Sort all nodes v of D's connected components D' ⊆ D with |D_i| ≥ γ in descending order of Ie(v, S) and mark them as unvisited
G' ← ∅
for each v ∈ D do
    if v is unvisited then
        Add u into G' and mark v as visited
    compute D-core D' of G'
    for each connected component D'_i of D' do
        if |D'_i| ≥ γ ∧ Ie(D'_i, S) > Ie(MIC, S) then
            MIC ← D'_i
Return MIC
```

Time Complexity. Following the time complexity analyzes of Algorithm 2, after adding each node u into G', the complexity of searching for the MIC is $O(|V_G| + |E_G|)$. Therefore, the total time complexity of Algorithm 5 is $O(|V_G| \cdot (|V_G| + |E_G|))$.

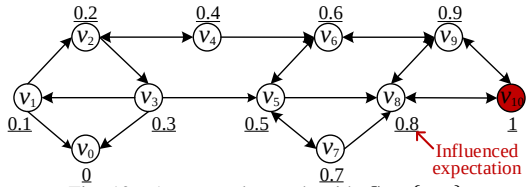


Fig. 10. An example graph with $S = \{v_{10}\}$

B. PROOF OF Theorem 5

Proof. Consider the graph in Figure 10. Let $\gamma = 2$; $k = 1$; $l = 1$.

Non-monotonicity. Let H_1 be the subgraph induced by vertices $\{v_6, v_9\}$; H_2 be the subgraph induced by vertices $\{v_5, v_6, v_9\}$; H_3 be the subgraph induced by vertices $\{v_5, v_6\}$. We have the following inequalities: (1) $Ie(H_1, v_{10}) = (0.6 + 0.9)/2 = 0.75 > Ie(H_2, v_{10}) = (0.5 + 0.6 + 0.9)/3 = 0.67$, and (2) $Ie(H_3, v_{10}) = 0.55 < Ie(H_2, v_{10}) = 0.67$. Obviously, $Ie(H, S)$ is non-monotonic.

Non-submodularity. For two arbitrary sets X and Y, if a function $f(\cdot)$ is submodular, it must satisfy $f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y)$. Let H_4 be the subgraph induced by vertices $\{v_8, v_9, v_{10}\}$. Then, $Ie(H_4, v_{10}) = 0.9$; $Ie(H_2 \cup H_4, v_{10}) = 0.76$; $Ie(H_2 \cap H_4, v_{10}) = 0.9$. We have the following inequality: $Ie(H_2, v_{10}) + Ie(H_4, v_{10}) = 0.67 + 0.9 = 1.57 < Ie(H_2 \cup H_4, v_{10}) + Ie(H_2 \cap H_4, v_{10}) = 0.76 + 0.9 = 1.66$. Therefore, $Ie(H, S)$ is non-submodular. □

C. PROOF OF Theorem 6

Proof. Applying chernoff inequalities, we have $\Pr[\tilde{Ie}(v, S) \cdot |\mathcal{I}| - |\mathcal{I}| \cdot Ie(v, S) \geq \epsilon |\mathcal{I}| \cdot Ie(v, S)] \leq \exp\left(-\frac{\epsilon^2}{2+\epsilon} \cdot |\mathcal{I}| \cdot Ie(v, S)\right)$. Given $|\mathcal{I}| \geq \frac{l \cdot (2+\epsilon) \cdot \log n}{\epsilon^2 \cdot Ie(v, S)}$, we have $\Pr[\tilde{Ie}(v, S) - Ie(v, S) \geq \epsilon \cdot Ie(v, S)] \leq n^{-l}$. Thus, $|\tilde{Ie}(v, S) - Ie(v, S)| < \epsilon \cdot Ie(v, S)$ holds with probability at least $1 - n^{-l}$. □

TABLE VI
PERFORMANCE OF PREDICTING MODELS (SKEWED SEED NODE SETS)

Model	EmailCore						WikiVote						Epinions					
	T-time	P-time	MSE	RMSE	MAE	R-Squared	T-time	P-time	MSE	RMSE	MAE	R-Squared	T-time	P-time	MSE	RMSE	MAE	R-Squared
RF	0.0573s	0.0014s	**0.0010**	**0.0316**	**0.0154**	**0.9752**	0.2743s	0.0018s	**0.0001**	**0.0117**	**0.0029**	**0.9938**	2.5600s	0.0047s	**0.0002**	**0.0164**	**0.0049**	**0.9943**
DT	0.0015s	**0.0001s**	0.0106	0.1030	0.0352	0.7375	0.0077s	**0.0001s**	0.0007	0.0282	0.0066	0.9645	0.0815s	**0.0002s**	0.0019	0.0446	0.0105	0.9580
KNN	**0.0003s**	0.0006s	0.0034	0.0589	0.0213	0.9141	**0.0004s**	0.0008s	0.0003	0.0198	0.0046	0.9825	**0.0015s**	0.0028s	0.0008	0.0283	0.0071	0.9831
Model	**Pokec**						**Livejournal**						**WikiLink**					
	T-time	**P-time**	**MSE**	**RMSE**	**MAE**	**R-Squared**	**T-time**	**P-time**	**MSE**	**RMSE**	**MAE**	**R-Squared**	**T-time**	**P-time**	**MSE**	**RMSE**	**MAE**	**R-Squared**
RF	701.65s	0.0964s	**6.29E-05**	**0.0079**	**0.0031**	**0.9909**	2241.8s	0.5089s	**5.26E-05**	**0.0072**	**0.0023**	**0.9891**	6309.8s	1.4454s	**6.47E-05**	**0.0080**	**0.0018**	**0.9919**
DT	21.906s	**0.0024s**	0.0005	0.0229	0.0093	0.9238	70.219s	**0.0119s**	0.0003	0.0198	0.0064	0.9186	200.98s	**0.0393s**	0.0004	0.0209	0.0044	0.9452
KNN	**0.0433s**	0.0535s	0.0002	0.0139	0.0052	0.9720	**0.1142s**	0.1664s	0.0001	0.0122	0.0036	0.9692	**0.1715s**	0.4785s	0.0002	0.0150	0.0027	0.9716

TABLE VII
PERFORMANCE OF PREDICTING MODELS (RANDOM SEED NODE SETS)

Model	Epinions						Slashdot						Livejournal					
	T-time	P-time	MSE	RMSE	MAE	R-Squared	T-time	P-time	MSE	RMSE	MAE	R-Squared	T-time	P-time	MSE	RMSE	MAE	R-Squared
RF	4.0114s	0.0043s	**2.57E-05**	**0.0050**	**0.0006**	**0.9751**	4.4609s	0.0046s	**3.73E-05**	**0.0061**	**0.0009**	**0.9891**	10729.8s	0.5085s	**1.95E-05**	**0.0044**	**0.0002**	**0.8682**
DT	0.1152s	**0.0002s**	0.0017	0.0423	0.0059	-0.7350	0.1282s	**0.0002s**	0.0016	0.0400	0.0044	-0.7656	476.21s	**0.0121s**	0.0002	0.0162	0.0010	-0.7709
KNN	**0.0015s**	0.0028s	0.0016	0.0410	0.0058	-0.6261	**0.0016s**	0.0029s	0.0014	0.0383	0.0042	-0.6189	**0.1139s**	0.1628s	0.0001	0.0130	0.0008	-0.1363

D. PROOF OF Theorem 7

Proof. As $Ie(D_1, S) > \max_{v \in V_{D_2}} Ie(v, S)$ and $D_3 \subseteq D_2$, it holds that $Ie(D_1, S) > \max_{v \in V_{D_3}} Ie(v, S)$. Thus, $Ie(D_1, S) > Ie(D_3, S)$. □

E. Full Experimental Results

This section shows the complete experiment results that are omitted in Section VII due to space constraints.

Performance of predictive models. In this experiment, we compare RF, DT, and KNN model on skewed seed node sets and random seed node sets. For each model, we report both *training time* (T-time) and *prediction time* (P-time), as well as *predictive accuracy* using four metrics: *Mean Squared Error* (MSE), *Root Mean Squared Error* (RMSE), *Mean Absolute Error* (MAE), and *R-squared*. For MSE, RMSE, and MAE, lower values indicate better performance. For R-squared, higher values are preferred. The results, summarized in Table VI and Table VII, show that, for both skewed seed node sets and random seed node sets, RF delivers the best predictive accuracy among the three models, with the smallest disparity between predicted and actual influenced expectations.

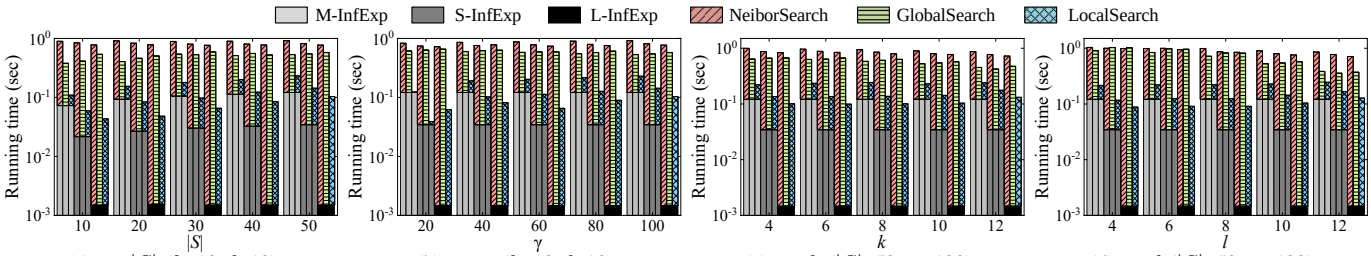
Effect of $|S|$. We analyze how the number of seed nodes $|S|$ affects running time. Results are shown in Figures 11(a)-18(a). We observe that the running times of M-InfExp and S-InfExp increase with $|S|$, as they need to generate more influence propagation instances. Therefore, M-InfExp and S-InfExp takes more time to generate influence propagation instances, resulting in higher running time. In addition, it can be seen that S-InfExp runs faster than M-InfExp because S-InfExp skips certain nodes when sampling influence propagation instances, thereby improving sampling efficiency. L-InfExp, on the other hand, maintains a relatively stable and lower running time. This is because, for online influenced expectation prediction, S-InfExp encodes S into a multi-hot vector $\boldsymbol{x}_S \in \{0,1\}^n$, so its performance is less sensitive to the changes in $|S|$. Moreover, the running times of NeiborSearch, GlobalSearch, and LocalSearch remain relatively stable with $|S|$ since this parameter mainly influences the computation of influenced expectations. LocalSearch is significantly faster than GlobalSearch and NeiborSearch across all settings, representing the high efficiency of our LocalSearch.

Effect of γ. We investigate the impact of the size constraint γ on algorithm performance. Figures 11(b)-18(b) show the results. We can observe that increasing γ has little effect on running time across all algorithms for influenced expectation computation. This is because γ only affects the size of the returned community, not the computation of influenced expectations. Moreover, the running time of LocalSearch decreases as the γ increases. This is because a larger γ leads to newly generated connected components that are less likely to satisfy the size condition, thereby reducing the number of communities that need to be examined. However, both NeiborSearch and LocalSearch show a slight increase in running time with larger γ. This is due to the fact that in the SearchLocalMIC algorithm, more vertices need to be inserted to search for the local most influenced community that meets the larger size condition, resulting in increased time required to compute the D-core and traverse all connected components.

Effect of k and l. We evaluate the effects of parameters k and l on algorithm performance. Figures 11(c)-18(c) present the results for k. Figures 11(d)-18(d) present the results for l. As k (or l) increases, the running time for NeiborSearch, GlobalSearch, and LocalSearch decreases because a larger k results in a smaller D-core, reducing the number of nodes that need to be processed. The running times of M-InfExp, S-InfExp, and L-InfExp remain unchanged as k and l only impact the community search.

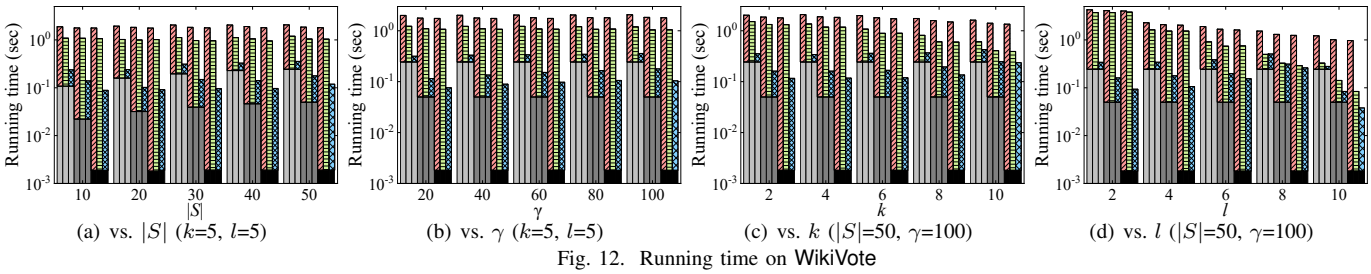
Scalability. We assess the scalability of our algorithms by varying the graph size. To this end, we randomly select a certain number of nodes from the graph to form the induced subgraph and test the running time of all algorithms. Figure 19 illustrates that the running times of M-InfExp, S-InfExp, NeiborSearch, GlobalSearch, and LocalSearch increase as the graph size grows. This is because NeiborSearch, GlobalSearch, and LocalSearch require more time to identify communities in larger graphs. For M-InfExp and S-InfExp, the increased running time is mainly due to the larger size of influence propagation instances. The running time of L-InfExp remains stable, as the efficiency of this learning based algorithm is not sensitive to the graph size.

Finally, we provide some experimental results on random seed node sets.



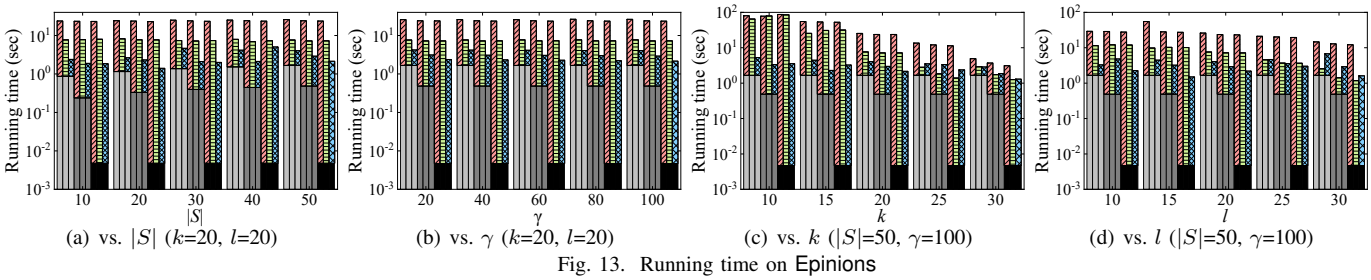
(a) vs. $|S|$ (k=10, l=10) (b) vs. γ (k=10, l=10) (c) vs. k ($|S|$=50, γ=100) (d) vs. l ($|S|$=50, γ=100)

Fig. 11. Running time on EmailCore



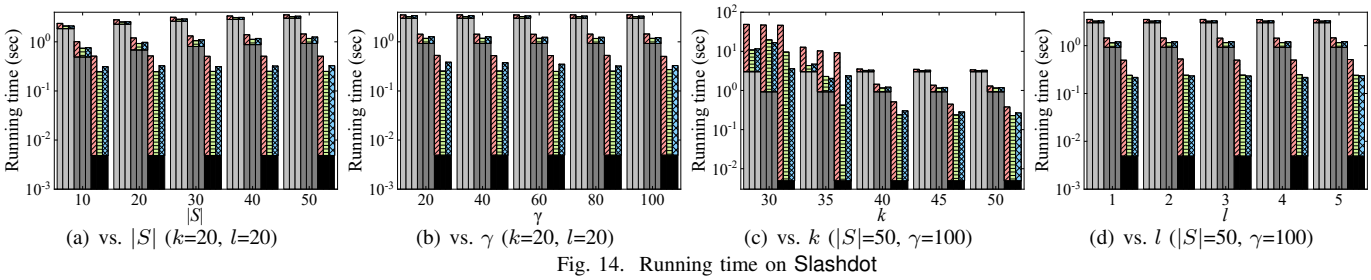
(a) vs. $|S|$ (k=5, l=5) (b) vs. γ (k=5, l=5) (c) vs. k ($|S|$=50, γ=100) (d) vs. l ($|S|$=50, γ=100)

Fig. 12. Running time on WikiVote



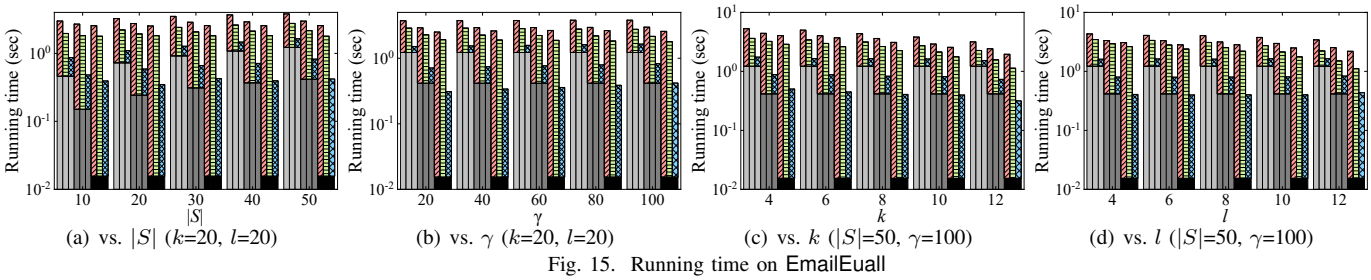
(a) vs. $|S|$ (k=20, l=20) (b) vs. γ (k=20, l=20) (c) vs. k ($|S|$=50, γ=100) (d) vs. l ($|S|$=50, γ=100)

Fig. 13. Running time on Epinions



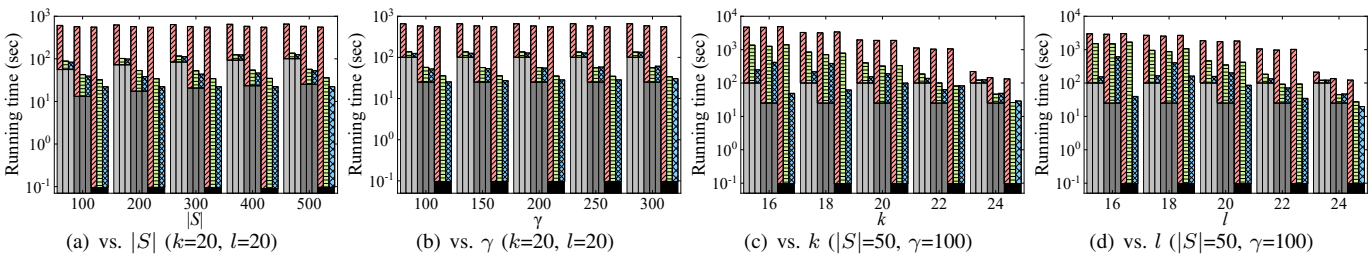
(a) vs. $|S|$ (k=20, l=20) (b) vs. γ (k=20, l=20) (c) vs. k ($|S|$=50, γ=100) (d) vs. l ($|S|$=50, γ=100)

Fig. 14. Running time on Slashdot



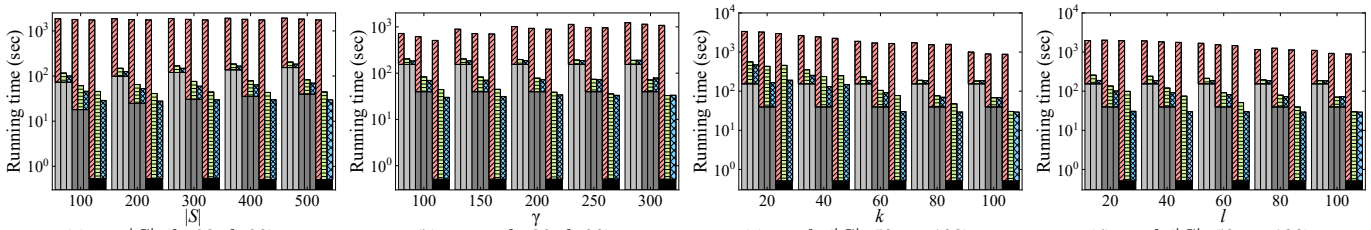
(a) vs. $|S|$ (k=20, l=20) (b) vs. γ (k=20, l=20) (c) vs. k ($|S|$=50, γ=100) (d) vs. l ($|S|$=50, γ=100)

Fig. 15. Running time on EmailEuall



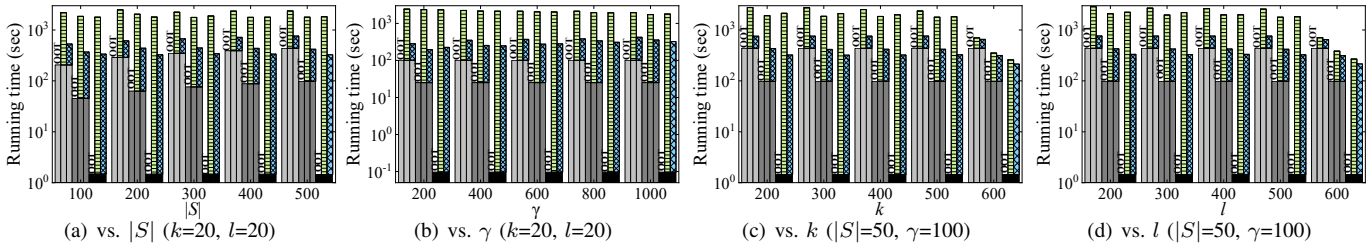
(a) vs. $|S|$ (k=20, l=20) (b) vs. γ (k=20, l=20) (c) vs. k ($|S|$=50, γ=100) (d) vs. l ($|S|$=50, γ=100)

Fig. 16. Running time on Pokec



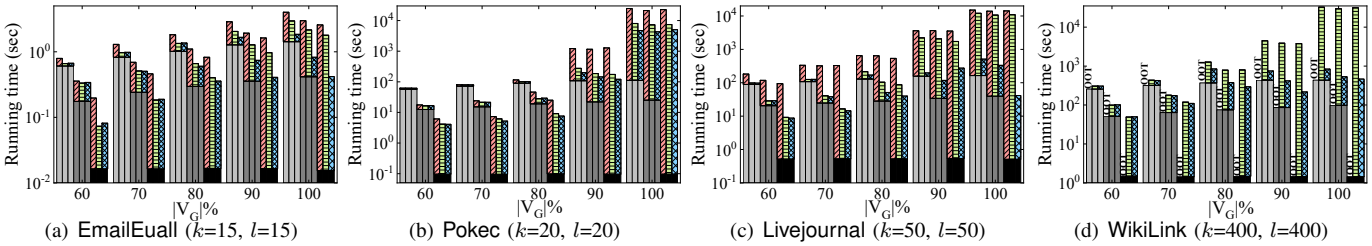
(a) vs. $|S|$ (k=20, l=20) (b) vs. γ (k=20, l=20) (c) vs. k ($|S|$=50, γ=100) (d) vs. l ($|S|$=50, γ=100)

Fig. 17. Running time on Livejournal



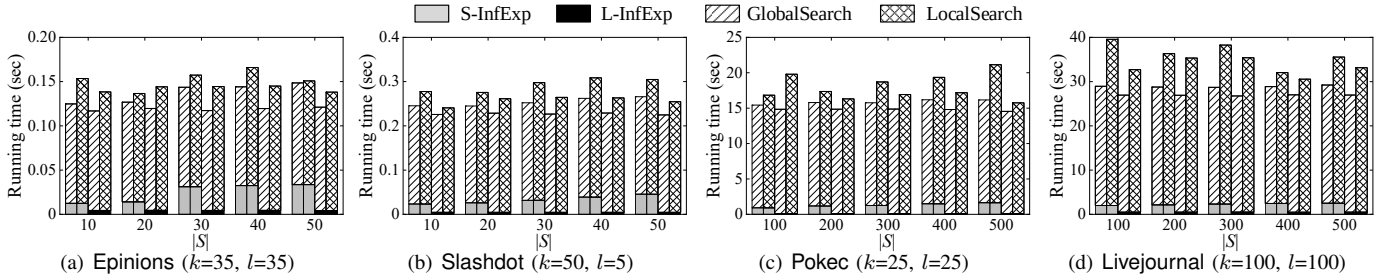
(a) vs. $|S|$ (k=20, l=20) (b) vs. γ (k=20, l=20) (c) vs. k ($|S|$=50, γ=100) (d) vs. l ($|S|$=50, γ=100)

Fig. 18. Running time on WikiLink



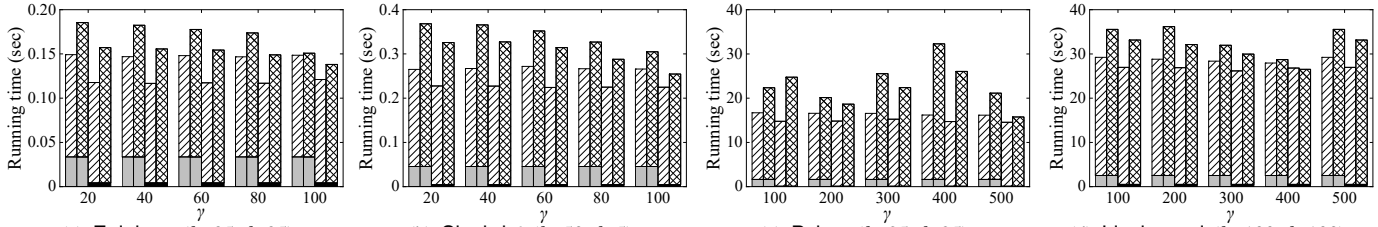
(a) EmailEuall (k=15, l=15) (b) Pokec (k=20, l=20) (c) Livejournal (k=50, l=50) (d) WikiLink (k=400, l=400)

Fig. 19. Scalability testing



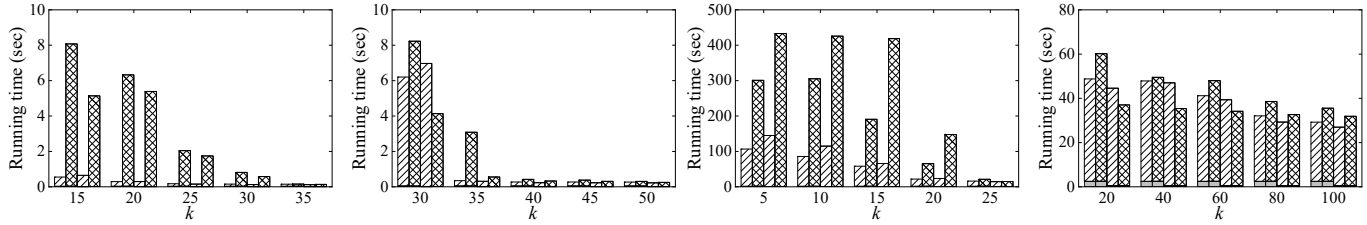
(a) Epinions (k=35, l=35) (b) Slashdot (k=50, l=5) (c) Pokec (k=25, l=25) (d) Livejournal (k=100, l=100)

Fig. 20. Running time vs. $|S|$ (Random seed node sets)



(a) Epinions (k=35, l=35) (b) Slashdot (k=50, l=5) (c) Pokec (k=25, l=25) (d) Livejournal (k=100, l=100)

Fig. 21. Running time vs. γ (Random seed node sets)



(a) Epinions ($|S|$=50, γ=100) (b) Slashdot ($|S|$=50, γ=100) (c) Pokec ($|S|$=500, γ=100) (d) Livejournal ($|S|$=500, γ=100)

Fig. 22. Running time vs. k (Random seed node sets)