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,qing-prof.li@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. This problem has practical applications in areas such as product promotion and epidemic control. We firstly formally define the MICS problem and prove its **NP**-hardness, demonstrating that no constant-factor approximation is feasible under the *avg* function. 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 develop a sampling-based algorithm, S-InfExp, which effectively estimates the influenced expectations with theoretical guarantee through influence propagation sampling. To enhance the efficiency, we also develop a learning-based approach, L-InfExp, which predicts the influence expectations for various seed node sets more swiftly. In the second phase, we focus on identifying the most influenced community based on the computed influenced expectations. We present two distinct algorithms for this purpose: GlobalSearch and LocalSearch. GlobalSearch operates in a top-down approach, iteratively removing nodes with the minimum influenced expectation, while LocalSearch employs a bottom-up strategy, determining the local most influenced community for each node. Our extensive experimental evaluations highlight two key findings: (1) L-InfExp achieves up to 1-2 orders of magnitude faster than S-InfExp, while maintaining comparable accuracy, and (2) both GlobalSearch and LocalSearch effectively identify the community with the highest influenced expectations.

PVLDB Reference Format:
Xueqin Chang, Qing Liu, Yunjun Gao, Baihua Zheng, Yi Cai, Qing Li. The Most Influenced Community Search on Social Networks. PVLDB, 18(1): XXX-XXX, 2025. doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/ZJU-DAILY/MICS.

1 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 [3, 35, 55, 59]. Researchers and industry professionals have focused on various aspects of influence analysis, such as predicting influence cascades [29, 56, 57, 59, 60], identifying influential nodes [3, 20, 21, 45, 52, 58, 62], and maximizing influence spread [5, 18, 19, 35, 47–49].

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, resource allocation, and personalized recommendation. For example, consider a scenario depicted in Figure 1(a) where a company employs a user v_5 to promote a new product. As v_5's influence spreads through the social network, other users are affected to varying extents, as shown in Figure 1(b). Among them, the users $\{v_2, v_3, v_5, v_6, v_9, v_{10}\}$ form a densely connected subgraph. This subgraph represents a community that is not only highly influenced by v_5 but also shares common interests. Targeting such a community can be more effective for marketing and advertising, as they are likely to be more receptive to the product. We refer to this subgraph as the *most influenced community*.

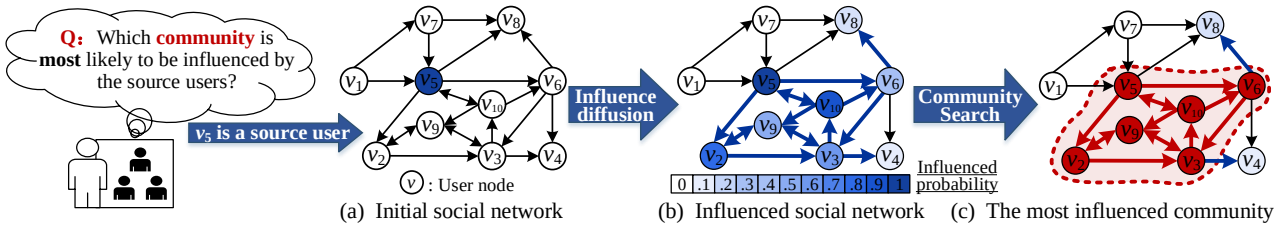
The Most Influenced Community Search Problem. Based on the above motivation, in this paper, we study the Most Influenced Community Search (MICS) problem. Specifically, given a directed graph $G(V_G, E_G)$, a size constraint γ, and a set of seed nodes $S \subset V_G$, our goal is to find a subgraph $H \subseteq G$ with $|H| \geq \gamma$ that is most influenced by S and is densely connected. To measure the degree to which the community is affected by S, we define the influenced expectation of a graph as the aggregate influence of the nodes within it, measured by the average[1] of their individual influenced expectations. Additionally, we employ D-core (a.k.a. (k, l)-core) [16] to assess the structural cohesiveness of the community.

The MICS problem has numerous practical applications, with two notable examples provided below. By addressing these real-world challenges, our study aims to provide valuable insights and tools for better decision-making in fields such as marketing and public health.

Application 1. *During a promotional campaign, companies often need to identify the most receptive user groups for distributing free products or coupons. To effectively promote a new product, they may first engage various types of influencers, such as social media influencers, celebrity influencers, and brand ambassadors. To craft the best marketing strategy and target the most receptive audience, companies can then apply MICS to pinpoint the community most influenced by these influencers, where users have the highest interest in their products, thereby optimizing marketing efforts and reducing costs associated with ineffective outreach.*

Proceedings of the VLDB Endowment, Vol. 18, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

[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 [8].



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

Figure 1: A motivating example

Application 2. *In managing infectious disease outbreaks, understanding how viruses spread through social networks is vital. MICS can help identify the most influenced communities affected by an outbreak. Public health authorities, such as the Centers for Disease Control and Prevention (CDC), can then focus their preventive measures, like distributing vaccines or medicines, on these communities to effectively mitigate the spread of the disease.*

Challenges and Solutions. We prove that the MICS problem is **NP**-hard and cannot be approximated within a constant-factor using the *avg* function. To 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 node set S. This involves determining the likelihood that each node v will be affected by S. Exact computation of influenced expectations is #**P**-hard, as detailed in Section 3.2. Therefore, we turn to approximation methods. Existing techniques in influence analysis, such as Monte-Carlo Simulation [25] and Reverse Influence Sampling [5], are designed for a task that is different from ours. These methods estimate the number of nodes influenced by a seed node set in the social network and are therefore not directly applicable to our problem. Consequently, 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 approximates the influenced expectation of each node with theoretical guarantees. The algorithm samples multiple *influence propagation instances* from G to generate an unbiased estimate for each node's influenced expectation. To improve sampling efficiency, we employ *subset sampling* technique that reduces the need to traverse all nodes. Moreover, we develop L-InfExp, a learning-based algorithm to predict the influenced expectation for each node. This method involves generating multiple seed node sets, computing their corresponding influenced expectations, and using these results as training and testing datasets. A nonlinear regression model $\mathcal{M}$ is trained on these datasets. Given a new seed set S, we employ $\mathcal{M}$ to predict the influenced expectations of different nodes, thereby significantly speeding up the process.

The second phase is to identify the most influenced community based on influenced expectations computed in the first phase. This phase bears similarity to the *influential community search* problem [2, 10, 31, 32, 42], where the goal is to find a subgraph with the highest aggregated measures, i.e., predefined metrics including H-index, PageRank, and degree. However, unlike influential community search, which seeks a community that either does not exceed a specified size constraint or has no size constraint, our problem requires the community to meet or exceed the size constraint to ensure the interventions or messages have a meaningful and measurable impact in practical applications. Recent work by Peng et al [42] proposed a local algorithm that adds nodes in descending order of predefined measures. Directly applying this approach to our problem yields suboptimal communities due to the differing size constraints. To address this, we propose two new algorithms for the second phase: GlobalSearch and LocalSearch.

GlobalSearch uses a heuristic deletion method to iteratively remove the nodes with the smallest influenced expectations, aiming to uncover a community with a higher overall influenced expectation. LocalSearch sorts nodes by their influenced expectations in descending order. Starting with the node v 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, to predict the influenced expectations with high efficiency.
- We design GlobalSearch and LocalSearch algorithms to identify the most influenced community. Moreover, we introduce a series of pruning strategies to enhance search efficiency.
- We conduct extensive experiments on eight real networks, along with a case study, to demonstrate the effectiveness, efficiency, and scalability of the proposed algorithms.

Roadmap. Section 2 reviews the related work. Section 3 formally defines and analyzes the problem of MICS. Section 4 provides an overview of our proposed techniques. Section 5 presents S-InfExp and L-InfExp algorithms. Section 6 introduces GlobalSearch and LocalSearch algorithms. Experimental results are reported in Section 7. Finally, Section 8 concludes the paper.

2 Related Work

Community Search. Community search aims to find cohesive subgraphs containing a given set of query nodes [14]. This topic has been extensively studied across various types of graphs, including directed graphs [16, 36, 38], attributed graphs [9, 22, 39], uncertain graphs [4, 43], temporal graphs [34, 37, 44], and heterogeneous graphs [15, 23]. Recently, many learning-based algorithms have been proposed for community search [24, 50, 51], which do not require pre-defining the community structure.

The most closely related community search to our work is influential community search [22, 30–33, 40, 42, 53, 61], which aims to find a community having the greatest influence. In this context, the community's influence is aggregated from individual influences of its nodes. However, our problem differs from influential community search in two key aspects. First, in influential community search, the nodes' influences are typically computed 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 seed node sets. Second, the size constraints are different. Specifically, our problem and the influential community search impose different constraints on the community size - 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, necessitating the development of new algorithms.

Influence Analysis. Influence analysis has been extensively studied in social networks and includes topics such as influence cascade prediction [29, 56, 57, 59, 60], influential nodes identification [3, 20, 21, 45, 52, 58, 62], and influence maximization/minimization [5, 18, 19, 35, 47–49]. Specifically, influence cascade prediction is to predict the overall diffusion volume, identify future active nodes, and clarify the propagation relationship between active 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 or minimize 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, which represents a distinct perspective on analyzing influence in social networks compared to existing influence analysis methods.

3 Preliminaries

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

3.1 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 the structure density of the community due to its computation efficiency.

Definition 1. **(D-core) [16].** *Given a directed graph $G(V_G, E_G)$ and 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 $d_H^{out}(v) \geq l$.*

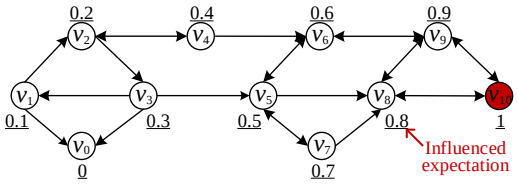


Figure 2: An example graph with $S = \{v_{10}\}$

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_8, v_9, and v_{10} is a (2,2)-core because $\forall v \in V_{H_1}$, $d_{H_1}^{in}(v) = d_{H_1}^{out}(v) = 2$, and we cannot reach another D-core by including any other vertex to H_1.

Influence propagation model. Given a seed node set, which may consist of both positive and negative nodes, we employ the well-known Independent Cascade (IC) model [25] to simulate influence propagation. In the IC model, for a given seed node set $S \subseteq V_G$, 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-neighbor $v \in N_G^{out}(u)$ 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 $Ie(v, S)$ as the *influenced expectation* of v, which represents the probability that v will be activated by S under the IC model. For example, in Figure 2, v_{10} is the seed node (i.e., $S = \{v_{10}\}$), and numbers next to vertices represent their influenced expectations under the current seed node set S. For example, $Ie(v_6, v_{10}) = 0.6$ indicates that there is a 60% chance of v_6 being activated by v_{10}. For a subgraph $H(V_H, E_H) \subseteq G$, we define the influenced expectation of H as follows.

Definition 2. **(Influenced expectation of H).** *Given a directed graph G, a subgraph $H(V_H, E_H) \subseteq G$, and a seed node set $S \subset V$, 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 2, $Ie(H, S)$ 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 [8].

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

Definition 3. **(MIC).** *Given a directed graph $G(V_G, E_G)$, a seed node set $S \subset V_G$, and two integers k and l, a subgraph $H(V_H, E_H) \subseteq G$ is an MIC if it satisfies the following conditions:*

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. Based on Definition 3, 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 is used to ensure that the community has a sufficiently large size in practical applications, such as marketing campaigns and political campaigns, where a larger community size is necessary to achieve effective outreach.

Example 1. *Figure 2 shows a directed graph $G(V_G, E_G)$. Let $S = \{v_{10}\}$, $\gamma = 2$, $k = 1$, and $l = 1$. The number next to each node represents its influenced expectation w.r.t., S. The directed subgraph H^* induced by nodes v_8, v_9, and v_{10} is a (1, 1)-core and has the maximum influenced expectation, i.e., $Ie(H^*, v_{10}) = (0.8 + 0.9 + 1)/3 = 0.9$. Hence, H^* is the result of MICS.*

3.2 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 $|MIC| \geq \gamma$ from G.*

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

Theorem 1. *The IEC problem 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 [11]. 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 [11], the IEC problem is also #P-hard. □

Theorem 1 shows the hardness of computing exact influenced expectation. Therefore, in Section 5, we develop 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 version of the Maximum Clique (DMC) problem to the CS problem. Specifically, given an undirected graph $G'(V', E')$ and an integer k', the DMC problem is to determine whether G' contains a k'-clique. Based on the DMC problem, 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', we add both (u, v) and (v, u) to E; (3) we add a new vertex $w \notin V'$ to V, and $\forall v \in V'$, we add both (w, v) and (v, w) to E; (4) we 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 problem efficiently. 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. Therefore, 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. [1], 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. Here, H must be a D-core with a size no smaller than γ.

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

Proof. Consider the graph in Figure 2. 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. □

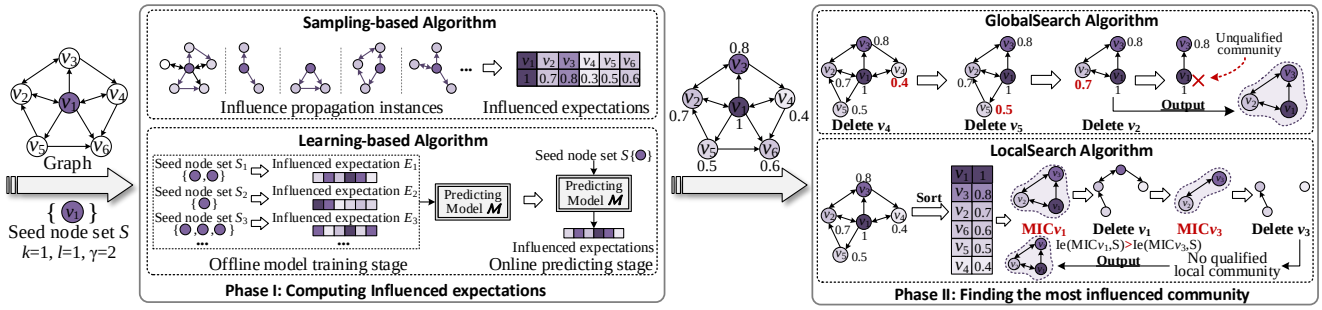


Figure 3: Framework overview

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.

4 Framework Overview

To efficiently 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 estimation algorithms. The first algorithm is to randomly sample a sufficient number of forward propagation instances to unbiasedly estimate the influenced expectation of each node. Based on the influenced expectation estimations obtained from the first algorithm, the second algorithm 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 GlobalSearch and LocalSearch algorithms. GlobalSearch algorithm finds the community in a top-down manner by iteratively removing the vertex with the smallest influenced expectation. In contrast, LocalSearch algorithm identifies the community in a bottom-up manner by finding the community containing vertex 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.

5 Influenced Expectation Computation

In this section, we first propose a sampling-based algorithm S-InfExp to approximately compute the influenced expectation of each node with theoretical guarantee. Based on S-InfExp, we introduce a learning-based method L-InfExp to predict the influenced expectations.

5.1 Sampling-based Algorithm

In Section 3.1, we introduced the IC model to simulate influence propagation. Building on this, we define the concept of influence propagation instance as follows.

Definition 4. **(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.

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

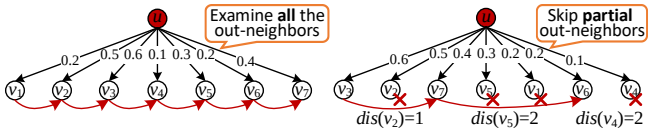
In other words, $Ie(v, S)$ is the sum of the existence probabilities of the influence propagation instances that include v. Computing $Ie(v, S)$ directly by enumerating all influence propagation instances is computationally prohibitive. Instead, we use sampling technique to sample partial influence propagation 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 seed nodes, 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



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

Figure 4: Illustrations of sampling methods

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 improve efficiency, we employ *subset sampling technique* [7], 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 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 [7], for a node u, subset sampling for discrete distributions achieves an expected time complexity of $O(1 + \kappa + \log d_G^{out}(u))$, where $\kappa = \sum_{v \in N_G^{out}(u)} w(u, v)$. This complexity is lower than $O(d_G^{out}(u))$. 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* [41] 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 $1 - \frac{1}{n}$.*

Proof. By applying chernoff inequalities, we have $\Pr[\tilde{Ie}(v, S) \cdot |\mathcal{I}| - |\mathcal{I}| \cdot Ie(v, S) \geq \epsilon \cdot |\mathcal{I}| \cdot Ie(v, S)] \leq \exp\left(-\frac{\epsilon^2}{2+\epsilon} \cdot |\mathcal{I}| \cdot Ie(v, S)\right)$. 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 $1 - n^{-l}$. □

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 $Ie(v, S)$. Following previous studies [17, 27, 54], we typically set the number of influence propagation instances $|\mathcal{I}|$ to the range of 10^3 to 10^4.

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

Algorithm 1 S-InfExp

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

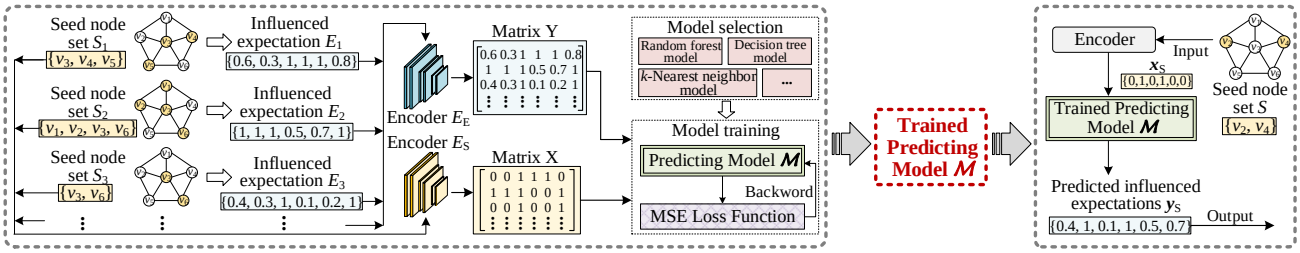
```
1: for each v ∈ V_G do
2:     I(v) ← 0;
3: for j ← 1 to α do
4:     Queue Q ← ∅; Mark ∀v ∈ V_G as unactivated;
5:     for each s ∈ S do
6:         Mark s as activated; Add s to Q;
7:         I(s) ← I(s) + 1;
8:     while Q ≠ ∅ do
9:         Pop u from Q;
10:        Sort u's out-neighbors in descending order of influence weight;
11:        v' ← The first out-neighbor of u;
12:        i ← 0;
13:        while i < d_G^out(u) do
14:            dist(v') ← ⌊log(rand())/log(1 − w(u, v'))⌋;
15:            i ← i + dist(v');
16:            if i ≥ d_G^out(u) then break;
17:            v'' ← the dist(v')-th out-neighbor after v;
18:            if rand() ≤ w(u, v'')/w(u, v') ∧ v'' is not activated then
19:                I(v'') ← I(v'') + 1;
20:                Insert v'' into Q; Mark v'' as activated;
21:            v' ← the node after v''; i++;
22: for each v ∈ V_G do
23:     Ĩe(v, S) ← I(v)/α;
24: Return ∪_{v∈V_G} Ĩe(v, S).
```

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).

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 $O(\alpha \cdot |V_G| \log(|V_G|))$.

5.2 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



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

Figure 5: The workflow of L-InfExp

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 prediction stage, for a given seed node 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. In what follows, we present a few key components 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. The general form of a nonlinear regression model is given below

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

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 [6], Decision Tree [13], and k-Nearest Neighbor models [12]. 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 correspondingly 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:

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

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.

6 MIC Search

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

6.1 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. This means that removing a vertex from D to form another D-core D' can either increase or decrease $Ie(D', S)$. Therefore, 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. Therefore, we use a greedy strategy 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

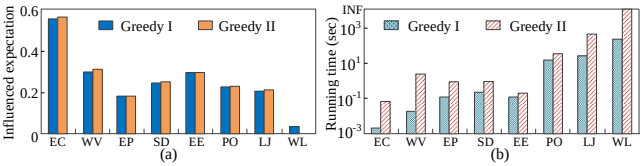


Figure 6: Greedy I vs. Greedy II

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 $Ie(D_1, S) > Ie(D_3, S)$.*

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

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 $O(|V_G| \cdot (|V_G| + |E_G|))$. Therefore, the overall time complexity of Algorithm 2 is $O(|V_G| \cdot (|V_G| + |E_G|))$.

6.2 Local Search Algorithm

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

Peng et al. [42] proposed a local algorithm for influential community search using the *Avg* function. The basic idea is to progressively add vertices to the community based on their descending influential scores. When applying this local algorithm directly to our problem,

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.

we can gradually add vertices to the community in descending order of their influenced expectations. 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. However, this method often yields a community with suboptimal influenced expectation, as demonstrated by experimental results in Section 7. 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.

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 of influenced expectations.

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: $\text{MIC}_v \leftarrow$ *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.

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 $Ie(v, S) \geq Ie(\text{MIC}_v, S)$. 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 minimize 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 terminated early. (iii) If MIC_v contains fewer than γ nodes, we can skip the D-core examination, as a valid D-core cannot be formed.

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 $|D_i| \geq \gamma$ in descending order of their 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 Algorithm 3 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*, which operates as follows. It 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). The function 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 satisfies the degree constraints and the size of $V_{G'}$ is at least γ, 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 unvisited neighbors of u by their influenced expectations in descending order and adds them to Q (Lines 13-16). The function terminates once MIC_v is found or the queue Q is empty.

Time Complexity. Following the time complexity analysis of Algorithm 2, Algorithm 4 has a worst-case time complexity of $O(|L_v|(|V_G| + |E_G|))$. This is because for each added node, the algorithm computes the D-core and traverses all connected components within it, which requires $O(|V_G| + |E_G|)$ time. With ω nodes having lower influenced expectation than the MIC, Algorithm 3 needs to compute the local most influenced community for $(|V_G| - \omega)$ nodes. Therefore, the overall time complexity of Algorithm 2 is $O(|L_v|(|V_G| - \omega)(|V_G| + |E_G|))$.

7 Experiments

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

Table 1: Dataset Statistics

Dataset	$n = \|V_G\|$	$m = \|E_G\|$	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

Our experiments utilize eight real-world directed networks, detailed in Table 1. The WikiLink dataset is from Konect [26], while the remaining networks are available on SNAP [28]. For each network, we apply the *weighted-cascade* model [25] to compute the influence weight $w(u, v)$ for each edge, defined as $w(u, v) = \frac{1}{|d_G^{in}(v)|}$.

Following the approach [46], we generate two types of seed node sets: *skewed* and *random*. The skewed seed nodes consist of the most influential nodes representing popular or influential users within the network. In contrast, the random seed nodes are sampled randomly from the network. Since the results from these two types of seed nodes are similar, we only present the results from the skewed seed node set for brevity. Detailed results from the random seed node set can be found in our technical report [8].

7.1 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) [6], Decision Tree (DT) [12], and k-Nearest Neighbor (KNN) [13]. 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 algorithm 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: *n_estimators* (*NE*)=100, *max_depth* (*MD*)=None, *min_samples_leaf* (*MSL*)=1, and *min_samples_split* (*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. We also evaluate RF with respect to hyperparameters effects, sensitivity analysis, and fit analysis.

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 2, 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 Random Forest. We study the impact of various hyperparameters combinations on RF performance as the number of seed nodes $|S|$ varies. The four hyperparameters examined include MD, MSL, MSS, and NE, with the following tested values: 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. MSE is used as the evaluation metric. The optimal hyperparameter combinations for different $|S|$ values across different datasets are presented in Table 3. It can be observed that, generally, smaller values of MSL and MSS lead to better model performance. This is because smaller values produce more complex decision trees, which fit the data more accurately. Conversely, MD and NE have a much smaller impact on the model's accuracy.

Exp-3: Sensitivity analysis of Random Forest. We conduct a sensitivity analysis of RF on Epinions dataset 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, detailed in Table 4, 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 values lead to lower prediction performance due to the simplified RF model. Conversely, increasing NE raises both training and prediction times due to the addition of more decision trees. Additionally, MD does not significantly impact RF's performance. Moreover, neither MD nor NE significantly affects the model's accuracy, while lower values of 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 Random Forest. We conduct a fit analysis experiment of the RF model on the Epinions dataset 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, suggesting stable model performance. Furthermore, a larger sample size improves model stability and generalization ability, as evidenced by lower testing errors.

7.2 Efficiency Evaluation

In this set of experiments, we evaluate the efficiency of our proposed algorithms. Due to space constraint and the similarity in experimental results, we present detailed results for the EmailEuall and Wiki-Link datasets in this paper. Results for other datasets can be found in [8].

Competitors. For influenced expectation computation, we propose sampling-based algorithm S-InfExp and learning-based algorithm L-InfExp. For MIC search, we design global search algorithm

Table 2: 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 3: 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 4: 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

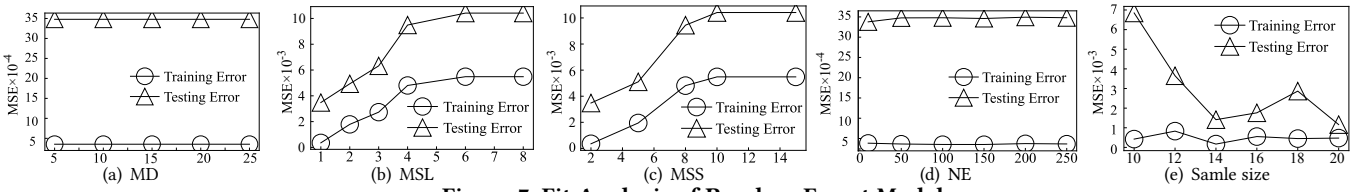


Figure 7: Fit Analysis of Random Forest Model

GlobalSearch and local search algorithm LocalSearch. By combination, we have four setting: **S**-InfExp+**G**lobalSearch, **S**-InfExp + **L**ocalSearch, **L**-InfExp+**G**lobalSearch, and **L**-InfExp + **L**ocalSearch.

Parameters. We test various parameters, including the integers k and l of D-core, the number of seed nodes $|S|$, the community size threshold γ, the graph size $|V_G|$, and the number of influence propagation instances $|\mathcal{I}|$. In each experiment, we vary only one parameter while keeping others at their default values. For small datasets such as EmailCore, WikiVote, Epinions, Slashdot, and EmailEuall, we have $|S| = 50$; for large datasets such as Pokec, LiveJournal, and WikiLink, we set $|S| = 500$. The default values of $|\mathcal{I}|$ and γ are 2000 and 100, respectively.

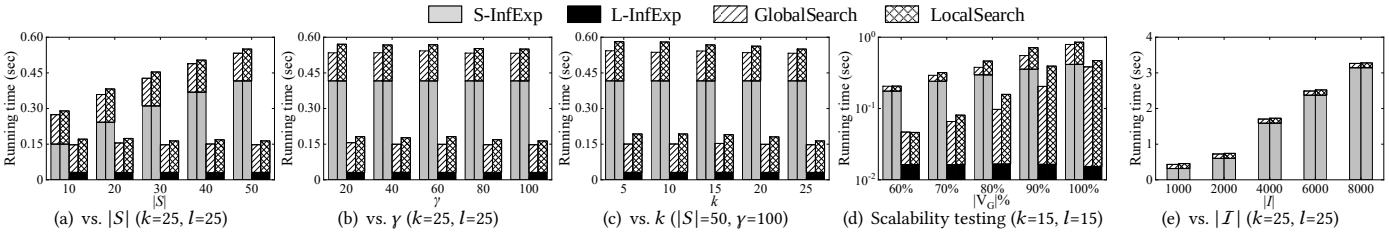
Exp-5: Effect of $|S|$. We analyze how the number of seed nodes $|S|$ affects running time. Results are shown in Figures 8(a) and 9(a). We observe that the running time of S-InfExp increases with $|S|$ due to the larger influence propagation instance it needs to handle. Therefore, S-InfExp takes more time to generate influence propagation instances, resulting in higher running time. 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 $|S|$. Additionally, the running times of both GlobalSearch and LocalSearch remain relatively stable with $|S|$ since this parameter mainly influences the computation of influenced expectations. Moreover, GlobalSearch performs better on small datasets, while LocalSearch is more efficient for larger datasets due to fewer nodes needing to be deleted in larger networks.

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. This is because γ only affects the size of the returned community, not the computation of influenced expectations. Moreover, LocalSearch shows a slight decrease in running time with larger γ due to increased pruning opportunities. When γ increases, the running time of LocalSearch slightly decreases because more nodes can be pruned during the search for the local most influenced community.

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 [8] because of space constraints. As k (or l) increases, the running time for 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 S-InfExp and L-InfExp remain unchanged as k and l only impact the community search, not the computation of influenced expectations.

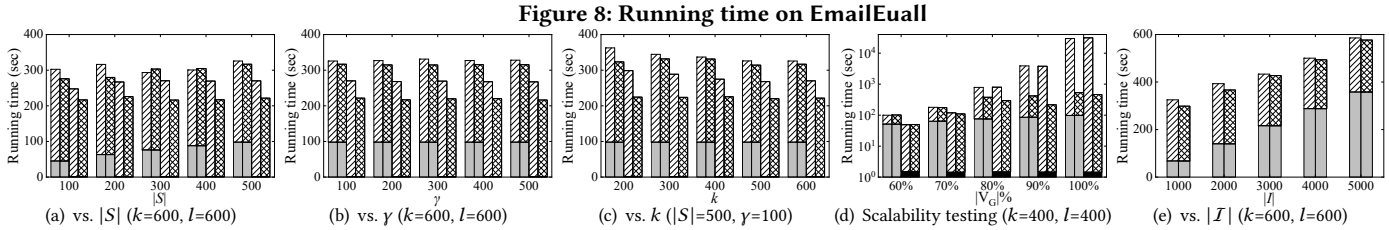
Exp-8: Scalability. We assess the scalability of our algorithms by varying the graph size. To this end, we randomly select a certain number of nodes from the graph to form the induced subgraph and test the running time of all algorithms. Figures 8(d) and 9(d) illustrate that the running time for S-InfExp, GlobalSearch, and LocalSearch increases with larger graphs. This is due to the greater time required for GlobalSearch and LocalSearch to find communities. S-InfExp experiences increased running time, primarily due to the increased number of influenced 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.

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 time of S-InfExp increases with $|\mathcal{I}|$, reflecting its time complexity. In contrast, GlobalSearch and LocalSearch show stable



(a) vs. $|S|$ (k=25, l=25) (b) vs. γ (k=25, l=25) (c) vs. k ($|S|$=50, γ=100) (d) Scalability testing (k=15, l=15) (e) vs. $|\mathcal{I}|$ (k=25, l=25)

Figure 8: Running time on EmailEuall



(a) vs. $|S|$ (k=600, l=600) (b) vs. γ (k=600, l=600) (c) vs. k ($|S|$=500, γ=100) (d) Scalability testing (k=400, l=400) (e) vs. $|\mathcal{I}|$ (k=600, l=600)

Figure 9: Running time on WikiLink

Table 5: Influenced Expectations Comparison

Dataset	Avg Function										Min Function			
	MIC						Original Graph		Maximal D-core		MIC			
	S+G	S+L	L+G	L+L	S+NL	L+NL	S-INF	L-INF	S-INF	L-INF	S+G	S+L	L+G	L+L
EC (k = 25, l = 25)	**0.5559**	0.5546	0.5470	0.5449	0.5459	0.5295	0.4121	0.4108	0.5459	0.5408	0.353	0.353	0.36	0.36
WV (k = 15, l = 10)	**0.3024**	0.2933	0.3017	0.2854	0.2761	0.2707	0.0803	0.0804	0.2761	0.2773	0.203	0.203	0.206	0.206
EP (k = 35, l = 35)	0.1846	**0.1851**	0.1822	0.1815	0.1825	0.1818	0.1409	0.1386	0.1815	0.1791	0.152	0.152	0.153	0.153
SD (k = 50, l = 5)	0.2463	0.2456	**0.2530**	0.2529	0.2376	0.2492	0.1813	0.1829	0.2456	0.2492	0.166	0.166	0.173	0.173
EE (k = 25, l = 25)	**0.2978**	0.2873	0.2879	0.2871	0.2833	0.2856	0.0618	0.0612	0.2534	0.249	0.048	0.048	0.046	0.046
PO (k = 25, l = 25)	**0.2287**	0.2138	0.2283	0.2159	0.2132	0.2019	0.0719	0.0713	0.1111	0.1111	0.154	0.154	0.153	0.153
LJ (k = 100, l = 100)	0.2083	**0.2085**	0.2041	0.2057	0.2042	0.2018	0.0375	0.0363	0.0246	0.0241	0.126	0.126	0.125	0.125
WL (k = 600, l = 600)	**0.0386**	0.0385	0.0379	0.0379	0.0258	0.0243	0.0298	0.0293	0.0121	0.0119	0.023	0.023	0.026	0.026

* S+G: S-InfExp+GlobalSearch, S+L: S-InfExp+LocalSearch, L+G: L-InfExp+GlobalSearch, L+L: L-InfExp+LocalSearch.

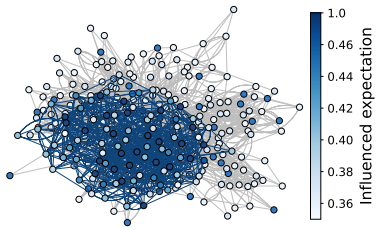


Figure 10: Case study on EmailCore

running times as $|\mathcal{I}|$ varies, since $|\mathcal{I}|$ mainly affects the computation of influenced expectations, which is independent of the MIC search handled by GlobalSearch and LocalSearch.

7.3 Effectiveness Evaluation

In this set of experiments, we evaluate the quality of returned communities by using different aggregation functions and conducting a case study.

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 set all parameters to their default values. The results are summarized in Table 5. For comparisons, we also report the influenced expectations for the entire graph and the maximal D-core. For function *avg*, we extend the algorithm proposed in [42] (denoted as NL) to find the community, using the combinations **S+NL** and **L+NL**, and report their influenced expectation. From Table 5, we make the following observations. (1) The MIC identified by the GlobalSearch algorithm has a slightly higher influenced expectation compared to the MIC found by LocalSearch. (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 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. (4) The influenced expectation of the MIC found by the NL method is lower than that found using our methods, demonstrating the superiority of our proposed algorithms. (5) 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.

Exp-11: Case study. Finally, we conduct a case study using EmailCore email communication network. We select 50 users as seed nodes and set $k = l = 25$, $\gamma = 100$. We use S-InfExp and GlobalSearch algorithms (i.e., S+G combination) to identify the community, which is visualized with blue edges in Figure 10. The color of node represents its influenced expectation, with darker color indicating higher expectations. The results reveal a dense community structure, where users within the community frequently interact with each other. Moreover, the influenced expectations of users within the community are notably higher than those outside the community.

8 Conclusion

In this paper, we introduce the most influenced community search (MICS) problem in social networks for the first time. To address this problem, we propose two algorithms: S-InfExp and L-InfExp. These algorithms compute the influenced expectation of each node using sampling and learning techniques, respectively. Additionally, we develop GlobalSearch and LocalSearch to identify the most influenced community based on these calculated influenced expectations. Specifically, GlobalSearch employs a top-down approach, while LocalSearch uses a bottom-up strategy to explore the community. Our extensive experiments on eight large real-world networks, coupled with a case study, demonstrate 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] Omid Amini, David Peleg, Stéphane Pérennes, Ignasi Sau, and Saket Saurabh. 2012. On the approximability of some degree-constrained subgraph problems. *Discrete Applied Mathematics* 160, 12 (2012), 1661–1679.

[2] Fei Bi, Lijun Chang, Xuemin Lin, and Wenjie Zhang. 2018. An optimal and progressive approach to online search of top-k influential communities. *Proceedings of the VLDB Endowment* 11, 9 (2018).

[3] Ranran Bian, Yun Sing Koh, Gillian Dobbie, and Anna Divoli. 2019. Identifying top-k nodes in social networks: a survey. *ACM Computing Surveys (CSUR)* 52, 1 (2019), 1–33.

[4] Francesco Bonchi, Francesco Gullo, Andreas Kaltenbrunner, and Yana Volkovich. 2014. Core decomposition of uncertain graphs. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1316–1325.

[5] Christian Borgs, Michael Brautbar, Jennifer Chayes, and Brendan Lucier. 2014. Maximizing social influence in nearly optimal time. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*. SIAM, 946–957.

[6] Leo Breiman. 2001. Random forests. *Machine learning* 45 (2001), 5–32.

[7] Karl Bringmann and Konstantinos Panagiotou. 2017. Efficient sampling methods for discrete distributions. *Algorithmica* 79 (2017), 484–508.

[8] Xueqin Chang, Qing Liu, Yunjun Gao, Baihua Zheng, Yi Cai, and Qing Li. 2024. The most influenced community search on social networks. https://github.com/ZJU-DAILY/MICS

[9] Lu Chen, Chengfei Liu, Rui Zhou, Jianxin Li, Xiaochun Yang, and Bin Wang. 2018. Maximum co-located community search in large scale social networks. *Proceedings of the VLDB Endowment* 11, 10 (2018), 1233–1246.

[10] Shu Chen, Ran Wei, Diana Popova, and Alex Thomo. 2016. 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*. 1553–1562.

[11] Wei Chen, Chi Wang, and Yajun Wang. 2010. 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*. 1029–1038.

[12] Vinícius G Costa and Carlos E Pedreira. 2023. Recent advances in decision trees: An updated survey. *Artificial Intelligence Review* 56, 5 (2023), 4765–4800.

[13] Padraig Cunningham and Sarah Jane Delany. 2021. K-nearest neighbour classifiers-a tutorial. *ACM computing surveys (CSUR)* 54, 6 (2021), 1–25.

[14] Yixiang Fang, Xin Huang, Lu Qin, Ying Zhang, Wenjie Zhang, Reynold Cheng, and Xuemin Lin. 2020. A survey of community search over big graphs. *The VLDB Journal* 29 (2020), 353–392.

[15] Yixiang Fang, Yixing Yang, Wenjie Zhang, Xuemin Lin, and Xin Cao. 2020. Effective and efficient community search over large heterogeneous information networks. *Proceedings of the VLDB Endowment* 13, 6 (2020), 854–867.

[16] Christos Giatsidis, Dimitrios M Thilikos, and Michalis Vazirgiannis. 2013. D-cores: measuring collaboration of directed graphs based on degeneracy. *Knowledge and information systems* 35, 2 (2013), 311–343.

[17] Amit Goyal, Wei Lu, and Laks VS Lakshmanan. 2011. Celf++ optimizing the greedy algorithm for influence maximization in social networks. In *Proceedings of the 20th international conference companion on World wide web*. 47–48.

[18] Qintian Guo, Chen Feng, Fangyuan Zhang, and Sibo Wang. 2023. Efficient algorithm for budgeted adaptive influence maximization: An incremental RR-set update approach. *Proceedings of the ACM on Management of Data* 1, 3 (2023), 1–26.

[19] Qintian Guo, Sibo Wang, Zhewei Wei, and Ming Chen. 2020. Influence maximization revisited: Efficient reverse reachable set generation with bound tightened. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2167–2181.

[20] Nesrine Hafiene, Wafa Karoui, and Lotfi Ben Romdhane. 2020. Influential nodes detection in dynamic social networks: A survey. *Expert Systems with Applications* 159 (2020), 113642.

[21] Shixun Huang, Zhifeng Bao, J Shane Culpepper, and Bang Zhang. 2019. Finding temporal influential users over evolving social networks. In *2019 IEEE 35th international conference on data engineering (ICDE)*. IEEE, 398–409.

[22] Xin Huang and Laks VS Lakshmanan. 2017. Attribute-driven community search. *Proceedings of the VLDB Endowment* 10, 9 (2017), 949–960.

[23] Yangqin Jiang, Yixiang Fang, Chenhao Ma, Xin Cao, and Chunshan Li. 2022. Effective community search over large star-schema heterogeneous information networks. *Proceedings of the VLDB Endowment* 15, 11 (2022), 2307–2320.

[24] Yuli Jiang, Yu Rong, Hong Cheng, Xin Huang, Kangfei Zhao, and Junzhou Huang. 2022. Query driven-graph neural networks for community search: from non-attributed, attributed, to interactive attributed. *Proceedings of the VLDB Endowment* 15, 6 (2022), 1243–1255.

[25] David Kempe, Jon Kleinberg, and Éva Tardos. 2003. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. 137–146.

[26] Jérôme Kunegis. 2013. Konect: the koblenz network collection. In *Proceedings of the 22nd international conference on world wide web*. 1343–1350.

[27] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. 2007. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. 420–429.

[28] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data.

[29] Huacheng Li, Chunhe Xia, Tianbo Wang, Sheng Wen, Chao Chen, and Yang Xiang. 2021. Capturing dynamics of information diffusion in SNS: A survey of methodology and techniques. *ACM Computing Surveys (CSUR)* 55, 1 (2021), 1–51.

[30] Rong-Hua Li, Lu Qin, Fanghua Ye, Guoren Wang, Jeffrey Xu Yu, Xiaokui Xiao, Nong Xiao, and Zibin Zheng. 2020. Finding skyline communities in multi-valued networks. *The VLDB Journal* 29 (2020), 1407–1432.

[31] Rong-Hua Li, Lu Qin, Fanghua Ye, Jeffrey Xu Yu, Xiaokui Xiao, Nong Xiao, and Zibin Zheng. 2018. Skyline community search in multi-valued networks. In *Proceedings of the 2018 International Conference on Management of Data*. 457–472.

[32] Rong-Hua Li, Lu Qin, Jeffrey Xu Yu, and Rui Mao. 2015. Influential community search in large networks. *Proceedings of the VLDB Endowment* 8, 5 (2015), 509–520.

[33] Rong-Hua Li, Lu Qin, Jeffrey Xu Yu, and Rui Mao. 2017. Finding influential communities in massive networks. *The VLDB Journal* 26 (2017), 751–776.

[34] Rong-Hua Li, Jiao Su, Lu Qin, Jeffrey Xu Yu, and Qiangqiang Dai. 2018. Persistent community search in temporal networks. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 797–808.

[35] Yuchen Li, Ju Fan, Yanhao Wang, and Kian-Lee Tan. 2018. Influence maximization on social graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering* 30, 10 (2018), 1852–1872.

[36] Xuankun Liao, Qing Liu, Jiaxin Jiang, Xin Huang, Jianliang Xu, and Byron Choi. 2022. Distributed D-core decomposition over large directed graphs. *Proceedings of the VLDB Endowment* 15, 8 (2022), 1546–1558.

[37] Longlong Lin, Pingpeng Yuan, Rong-Hua Li, Chunxue Zhu, Hongchao Qin, Hai Jin, and Tao Jia. 2024. QTCS: Efficient query-centered temporal community search. *Proceedings of the VLDB Endowment* 17, 6 (2024), 1187–1199.

[38] Qing Liu, Minjun Zhao, Xin Huang, Jianliang Xu, and Yunjun Gao. 2020. Truss-based community search over large directed graphs. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2183–2197.

[39] Qing Liu, Yifan Zhu, Minjun Zhao, Xin Huang, Jianliang Xu, and Yunjun Gao. 2020. VAC: vertex-centric attributed community search. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 937–948.

[40] Wensheng Luo, Xu Zhou, Jianye Yang, Peng Peng, Guoqing Xiao, and Yunjun Gao. 2020. Efficient approaches to top-r influential community search. *IEEE Internet of Things Journal* 8, 16 (2020), 12650–12657.

[41] Michael Mitzenmacher and Eli Upfal. 2017. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press.

[42] You Peng, Song Bian, Rui Li, Sibo Wang, and Jeffrey Xu Yu. 2022. Finding top-r influential communities under aggregation functions. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 1941–1954.

[43] You Peng, Ying Zhang, Wenjie Zhang, Xuemin Lin, and Lu Qin. 2018. Efficient probabilistic k-core computation on uncertain graphs. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 1192–1203.

[44] Hongchao Qin, Rong-Hua Li, Ye Yuan, Guoren Wang, Weihua Yang, and Lu Qin. 2020. Periodic communities mining in temporal networks: Concepts and algorithms. *IEEE Transactions on Knowledge and Data Engineering* 34, 8 (2020), 3927–3945.

[45] Fabián Riquelme and Pablo González-Cantergiani. 2016. Measuring user influence on Twitter: A survey. *Information processing & management* 52, 5 (2016), 949–975.

[46] Michael Simpson, Farnoosh Hashemi, and Laks VS Lakshmanan. 2022. Misinformation mitigation under differential propagation rates and temporal penalties. *Proceedings of the VLDB Endowment* 15, 10 (2022), 2216–2229.

[47] Jing Tang, Xueyan Tang, Xiaokui Xiao, and Junsong Yuan. 2018. Online processing algorithms for influence maximization. In *Proceedings of the 2018 ACM SIGMOD International Conference on Management of Data*. 991–1005.

[48] Youze Tang, Yanchen Shi, and Xiaokui Xiao. 2015. Influence maximization in near-linear time: A martingale approach. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. 1539–1554.

[49] Youze Tang, Xiaokui Xiao, and Yanchen Shi. 2014. Influence maximization: Near-optimal time complexity meets practical efficiency. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. 75–86.

[50] Jianwei Wang, Kai Wang, Xuemin Lin, Wenjie Zhang, and Ying Zhang. 2024. Efficient unsupervised community search with pre-trained graph transformer. *Proceedings of the VLDB Endowment* 17, 9 (2024), 2227–2240.

[51] Jianwei Wang, Kai Wang, Xuemin Lin, Wenjie Zhang, and Ying Zhang. 2024. Neural attributed community search at billion scale. *Proceedings of the ACM on Management of Data* 1, 4 (2024), 1–25.

[52] Yu Wang, Gao Cong, Guojie Song, and Kunqing Xie. 2010. 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*. 1039–1048.

[53] Yanping Wu, Jun Zhao, Renjie Sun, Chen Chen, and Xiaoyang Wang. 2021. Efficient personalized influential community search in large networks. *Data Science and Engineering* 6, 3 (2021), 310–322.

[54] Jiadong Xie, Fan Zhang, Kai Wang, Xuemin Lin, and Wenjie Zhang. 2023. Minimizing the influence of misinformation via vertex blocking. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 789–801.

[55] Yu Yang and Jian Pei. 2019. Influence analysis in evolving networks: A survey. *IEEE Transactions on Knowledge and Data Engineering* 33, 3 (2019), 1045–1063.

[56] Tauhid R Zaman, Ralf Herbrich, Jurgen Van Gael, and David Stern. 2010. Predicting information spreading in twitter. In *Workshop on computational social science and the wisdom of crowds, nips*, Vol. 104. Citeseer, 17599–601.

[57] Bolei Zhang, Zhuzhong Qian, and Sanglu Lu. 2016. Structure pattern analysis and cascade prediction in social networks. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD*. Springer, 524–539.

[58] Junzhou Zhao, Shuo Shang, Pinghui Wang, John CS Lui, and Xiangliang Zhang. 2019. Tracking influential nodes in time-decaying dynamic interaction networks. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 1106–1117.

[59] Fan Zhou, Xovee Xu, Goce Trajcevski, and Kunpeng Zhang. 2021. A survey of information cascade analysis: Models, predictions, and recent advances. *ACM Computing Surveys (CSUR)* 54, 2 (2021), 1–36.

[60] Fan Zhou, Xovee Xu, Kunpeng Zhang, Goce Trajcevski, and Ting Zhong. 2020. Variational information diffusion for probabilistic cascades prediction. In *IEEE INFOCOM 2020-IEEE conference on computer communications*. IEEE, 1618–1627.

[61] Yingli Zhou, Yixiang Fang, Wensheng Luo, and Yunming Ye. 2023. Influential community search over large heterogeneous information networks. *Proceedings of the VLDB Endowment* 16, 8 (2023), 2047–2060.

[62] Yuqing Zhu, Jing Tang, and Xueyan Tang. 2020. Pricing influential nodes in online social networks. *Proceedings of the VLDB Endowment* 13, 10 (2020), 1614–1627.

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 5. **(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 $Ie(H, S) = \min_{v \in H_o} Ie(v, S)$, which represents the probability that H is influenced by S under propagation model M.*

It can be verified that the MIC defined in Definition 3 is also suitable for the above definition. Then, we extend our GlobalSearch and LocalSearch algorithms 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 $O(|V_G| \cdot (|V_G| + |E_G|))$.

B Full Experimental Results

This section shows the complete experiment results that are omitted in Section 7 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 6 and Table 7, 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. Therefore, we employ RF in the L-InfExp algorithm due to its superior predictive accuracy.

Effect of $|S|$. We analyze the number of seed nodes $|S|$ on running time. Results are shown in Figures 16(a)-16(d). We observe that the running time of S-InfExp increases with $|S|$ due to the larger influence propagation instance it needs to handle. Therefore, S-InfExp takes more time to generate influence propagation instances, resulting in higher running time. 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 $|S|$. Additionally, the running times of both GlobalSearch and LocalSearch remain relatively stable with $|S|$ since this parameter mainly influences the computation of influenced expectations. Moreover, GlobalSearch performs better on small datasets, while LocalSearch is more efficient for larger datasets due to fewer nodes needing to be deleted in larger networks.

Effect of γ. We investigate the impact of the size constraint γ on algorithm performance. Figures 17(a)-17(d) show the results. We can observe that increasing γ has little effect on running time

Table 6: 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 7: 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

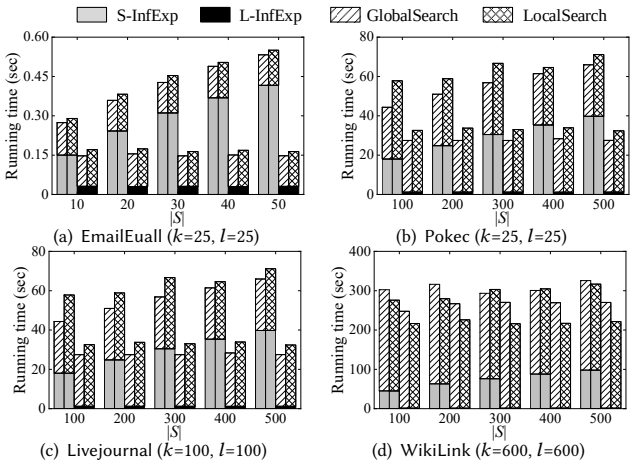


Figure 11: Running time vs. $|S|$

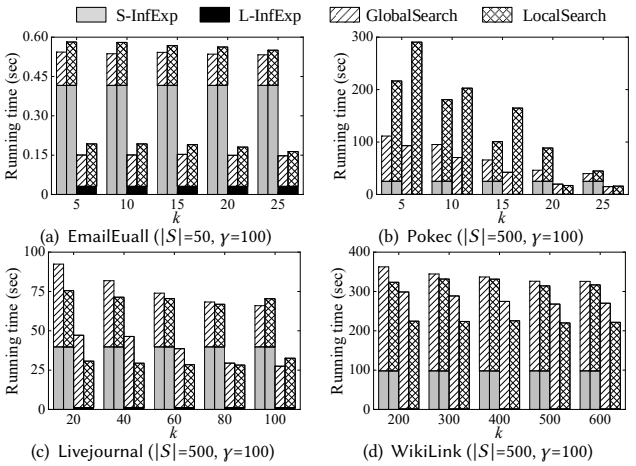


Figure 13: Running time vs. k

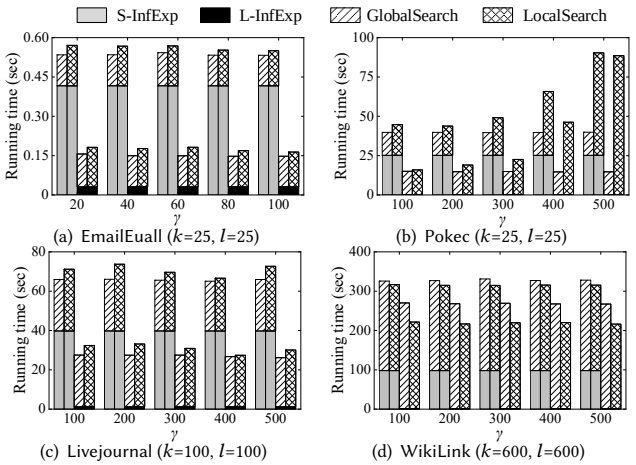


Figure 12: Running time vs. γ

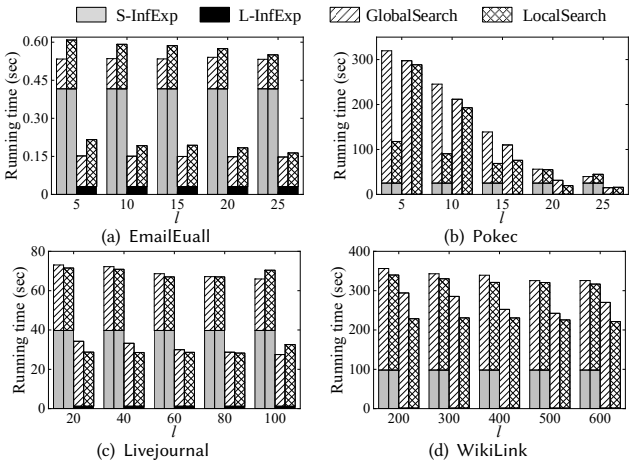


Figure 14: Running time vs. l

across all algorithms. This is because γ only affects the size of the returned community, not the computation of influenced expectations. Moreover, LocalSearch shows a slight decrease in running time with larger γ on most datasets due to increased pruning opportunities. When γ increases, the running time of LocalSearch slightly decreases because more nodes can be pruned during the search for the local most influenced community. As for Pokec, the runtime of LocalSearch increases as γ grows. This is because a larger subgraph is formed during the search for the most influenced local community, leading to more time required to compute the connected D-core.

Effect of k and l. We evaluate the effects of parameters k and l on algorithm performance. Figures 19(a)-19(d) present the results for k. Figures 14(a)-14(d) present the results for l. As k (or l) increases, the running time for GlobalSearch and LocalSearch decreases because a larger k results in a smaller D-core, reducing the number of nodes

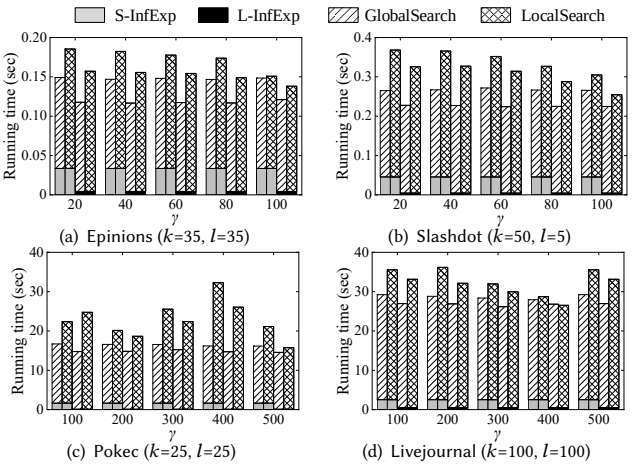


Figure 17: Running time vs. γ (Random seed node sets)

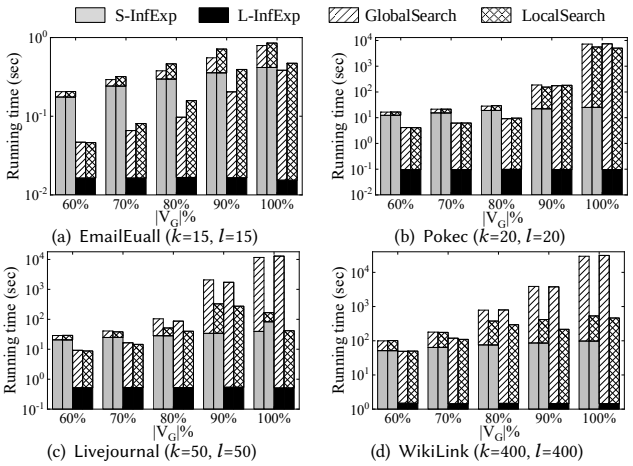


Figure 15: Scalability testing

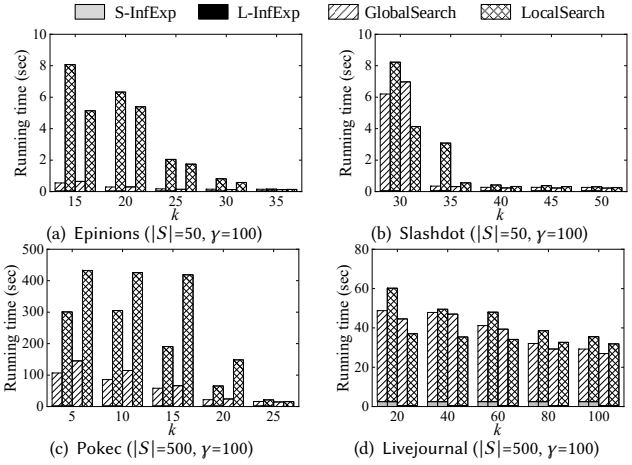


Figure 18: Running time vs. k (Random seed node sets)

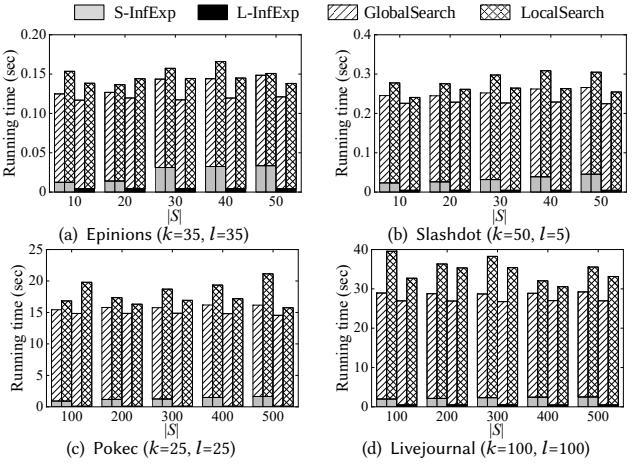


Figure 16: Running time vs. $|S|$ (Random seed node sets)

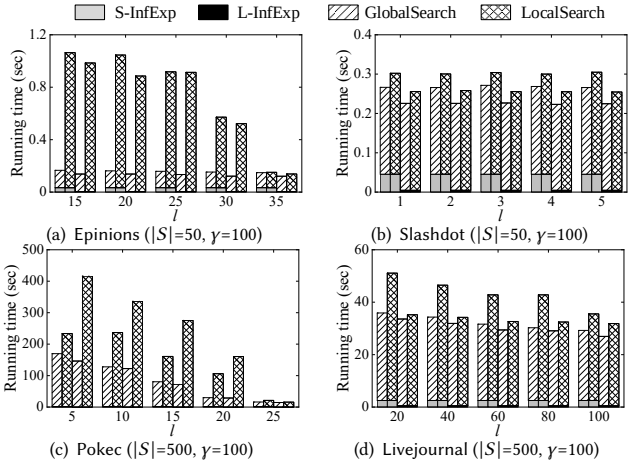


Figure 19: Running time vs. l (Random seed node sets)

that need to be processed. The running times of S-InfExp and L-InfExp remain unchanged as k and l only impact the community search, not the computation of influenced expectations.

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 15(a)-15(d) illustrate that the running time for S-InfExp, GlobalSearch, and LocalSearch increases with larger graphs. This is due to the greater time required for GlobalSearch and LocalSearch to find communities. S-InfExp experiences increased running time, primarily due to the increased number of influenced 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.