Chapter 4

Community-based Context-aware Influence Maximization

This Chapter investigates the role of contextual feature (topics) in effective seed selection on social networks and also utilize the community structure of network for efficient seed selection.

4.1 Introduction

Besides the inapplicability of complex diffusion models, the greedy algorithm embraces two major issues. The first issue is that the greedy algorithm is not time efficient. Therefore, it is not scalable with large networks. To alleviate this problem, several research works have been proposed such as heuristic metrics based [24–26], sub-modularity based [27–29], influence path based [30–32], community based [33, 34], etc.

These approaches are an order of magnitude faster than the traditional greedy approach, but the quality is still an issue. The second issue is the effectiveness of seed which limits the quality of seed. To alleviate this problem, several context-aware influence maximization techniques were introduced using some contextual features such as location-aware [16, 17], time-aware [18, 19], topic-aware [20, 21], and competitive [22, 23], etc. These context-aware approaches generate quality seed with less efficiency and scalability.

To address the above challenges, a community-based context-aware influence maximization (C2IM) algorithm is proposed. C2IM considers community structure and *user's interests* to find effective seed efficiently. The algorithm utilizes node degree distribution to identify the community structure of the social network. The users have their contextual features like *user's interest* (as topics) in the real-world social network. The users with similar interests (friends or friends of friends) have more influence on each other. For example, a cricket fan will be more influenced by a cricket player (cricket fan) rather than a soccer player (soccer fan). Therefore, the algorithm utilizes *user's interest* to find effective seed nodes. To address *user's interest* in the IM problem, C2IM models influence graph as a context-aware graph. For example, given an edge from node x to y with contextual distribution <cricket:0.7, politics:0.4, tech:0.5> indicates that node y influenced by x on topics cricket, politics, and health with probability 0.7, 0.4, and 0.5 respectively. There are many

real-world applications for context-aware IM such as context-aware rumor control and context-aware advertising. The main contributions of the chapter are as follows.

- Traditional diffusion models (IC,LT) are extended to context-aware independent cascade model (CIC) and context-aware linear threshold model (CLT) for influence spreading.
- Defined two quality measures: weighted fully connectivity (WFC) and percentage of missing links (PML) to analyze community structure.
- Exploring third objective, a novel algorithm C2IM is presented using community-based framework.
- The empirical results on six real-world networks show that the proposed algorithm C2IM performs better than the base method CIM in terms of influence spread with almost the same running time. C2IM is more time-efficient than base method TIM with approximate influence spread.

4.1.1 Data Model

Users in the real social network have their own interests and they likely have more influence on those friends who have similar interests. We consider an influence graph $G(V,E)$ where each edge $(u,v) \in E$, $u \in V$,

$v \in V$ is associated with interest distribution $I_{u,v} = (ip^1_{u,v}, ip^2_{u,v}, ip^3_{u,v}, \ldots, ip^t_{u,v})$ on t number of topics. $ip^i_{u,v}$ denotes the influence probability of user u on v under topic i. For each edge an interest distribution can be estimated using text-based topic discovery approaches [162].

4.1.2 Context-aware Information Diffusion Models

Traditional diffusion models (IC,LT) are extended and presented as CIC and CLD diffusion models. Initially, all nodes are inactive state and only active users u can propagate influence to their neighbors $N_{out}(u)$, similar to traditional models. For any contextual query Q, initially k seed users are selected and their state change from inactive to active.

Context-based linear threshold model. Similar to traditional threshold model, each user u has an activation threshold $\theta_u \in [0,1]$. An inactive user v becomes active only if $CI(N^A_{inc}(v)) \geq \theta_v$ is satisfied. The combined influence of active neighbors of v, $CI(N^A_{inc}(v))$ is estimated as follows.

$$CI(N^A_{inc}(v)) = \sum_{i=1}^{i=t} \sum_{u \in N_{inc}(v)} q_i . ip^i_{u,v} \tag{4.1}$$

where $q_i \in Q = \langle q_1, q_2, \ldots, q_t \rangle$.

Context-based independent cascade model. Similar to traditional independent cascade model, each active user u has only chance to activate

its out-going neighbors $v \in N_{out}(u)$ and after that, u stays active. User u succeeds to activate v with activation probability $ap_{u,v}$.

Definition 4.1.1. **(Activation probability)**. Given an edge $(u,v) \in E$ with influence probability for t topics, $I_{u,v} = (ip^1_{u,v}, ip^2_{u,v}, ip^3_{u,v}, \ldots, ip^t_{u,v})$ and query distribution $Q = \{q_1, q_2, \ldots, q_t\}$ under CIC diffusion model then node u activate v with activation probability $ap_{u,v}$, as follows.

$$ap_{u,v} = \sum_{i=1}^{t} ip^i_{u,v} \cdot q_i \tag{4.2}$$

In this chapter, CLT and CIC models are utilized for information diffusion.

Theorem 4.1. *The expected influence spread $\sigma(S|Q)$ with contextual query Q is sub-modular under CLT and CIC models.*

Proof. The proof of Theorem 4.1 is given in Appendix A.1. □

Theorem 4.2. *The community-based context-aware influence maximization under LT and IC influence diffusion model is NP-hard.*

Proof. The proof of Theorem 4.2 is given in Appendix A.1. □

4.2 Proposed Approach

In this section, a novel algorithm C2IM for maximizing influence spread in social networks is presented. The algorithm uses community-based

framework to identify seed nodes within communities using their local influence. It also considers *user's interests* to make seed more effective. The users who are more similar in their interest, they have more influence on each other. For example, if we want to influence people about some information which is closely related to mathematics and less related to politics, it will query something like Q=⟨math:0.8, politics:0.2⟩.

First, we give an overview of C2IM and then we explore algorithm in more detail.

4.2.1 Algorithm Overview

FIGURE 4.1 explains the framework of the proposed algorithm. C2IM contains the following steps:

1 **Preprocessing:** In this step, C2IM finds the non-overlapping community structure of graph using hierarchical clustering.

2 **Finding non-desirable nodes:** It selects r fraction of nodes in each community as non-desirable nodes.

3 **Selecting seed nodes:** Select $f\%$ of seeds from each communities based on their local influence and $(100-f)\%$ of seeds by considering whole graph as a single community based on their global influence.

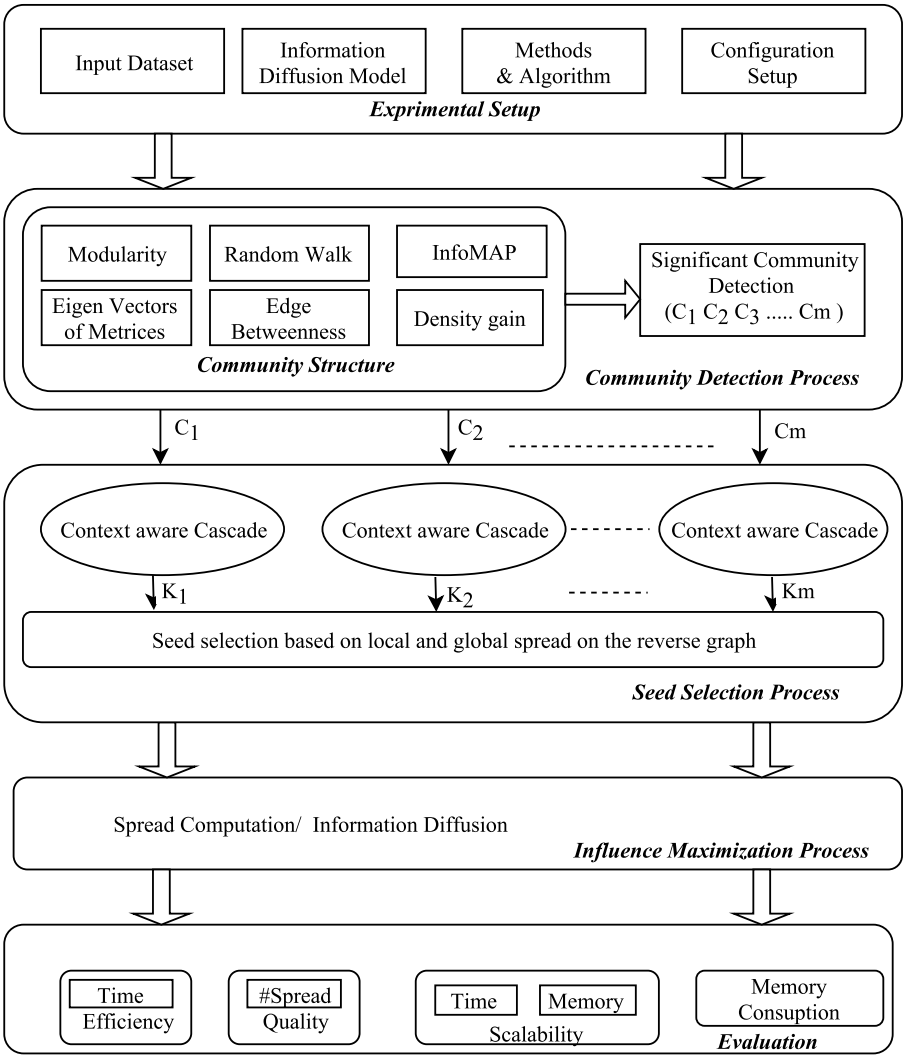
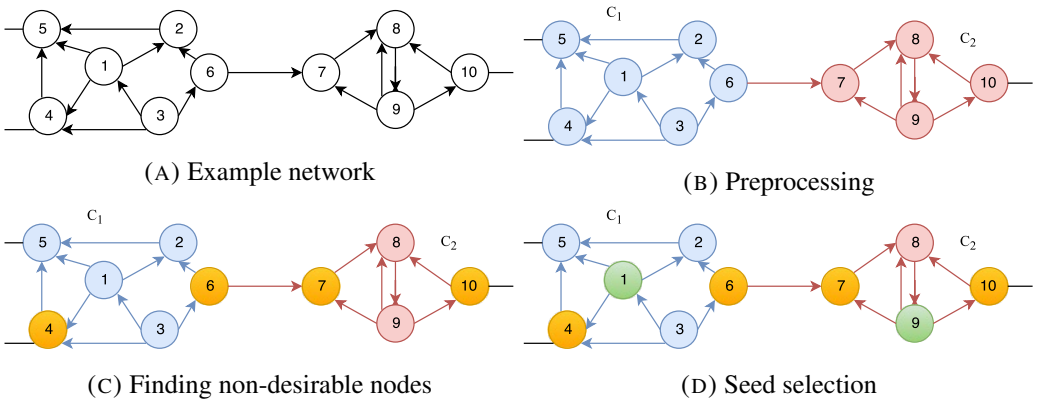


FIGURE 4.1: The Proposed C2IM Framework

4 **Computing spread of nodes:** It computes expected influence spread $\sigma(S|Q)$ with contextual query Q under diffusion models CLT and CIC.

FIGURE 4.2 gives an overview of C2IM using an example. FIGURE 4.2a shows an example graph. FIGURE 4.2b shows that graph partitioned into two communities C_1 and C_2, where each community node is shown in

different colors. FIGURE 4.2c identifies non-desirable nodes {4,6,7,10}. FIGURE 4.2d selects seed nodes {1,9}.



(A) Example network (B) Preprocessing

(C) Finding non-desirable nodes (D) Seed selection

FIGURE 4.2: The C2IM Overview using an Example Network

4.2.2 Algorithm Description

In this section, we describe the algorithm in detail.

4.2.2.1 Preprocessing

C2IM performs community detection process as preprocessing step. To find the crisp community structure C of graph G, we proposed an algorithm named CDA. Our algorithm uses hierarchical clustering approach based on degree distribution of the network. In this algorithm, we first begin by making each individual node as a single community. We define the density of a community as the root mean square (RMS) values of the degree of nodes in the community.

Definition 4.2.1. *(Density).* The density of a community $C_i = \{u_1, u_2, u_3, \ldots, u_m\}$ is root mean square values of degree of nodes of community C_i, i.e. density of C_i is given as.

$$D(C_i) = \sqrt{\frac{\sum_{u_j \in C_i} d(u_j)^2}{|C_i|}} \tag{4.3}$$

where $d(u_j)$ denotes the degree of node $u_j \in C_i$.

Now we merge the two communities only if relative change of density is above threshold Δ. More formally, let $D(C_{i,j})$ denotes the density of community formed by merging community C_i and C_j, then we will merge two communities C_i and C_j only if $\Delta_{C_i,C_j} > \Delta$. Density gain (Δ_{C_i,C_j}) is defined as follows.

$$\Delta_{C_i,C_j} = \frac{D(C_{i,j}) - (D(C_i) + D(C_j))}{D(C_{i,j})} \tag{4.4}$$

We control the structure of C using density gain measure. So we can set a threshold value Δ and decide how sparse or dense a community structure will be based on our requirement. Repeatedly merges the two communities with maximum density gain, until no further merge possible. Suppose that two merge communities $C_{i,j}$ and $C_{i,l}$ have equal maximum density gain. Then, we can decide which two will be merged based on their *GetCommunity* values. $GetCommunity(C_i, C_j)$ value is defined as follows.

Definition 4.2.2. *(GetCommunity).* $GetCommunity(C_i, C_j)$ finds the number of paths triplets (u,v,w) such that $(u,v) \in E$,$(w,v) \in E$, $u \in C_i$, $w \in C_j$,$v \notin C_i$ and $v \notin C_j$.

FIGURE 4.2a and 4.2b show the sample graph and their respective community structure respectively. We can check the quality of CDA using two new quality measures: *weighted fully connectivity* (WFC) and *percentage of missing links* (PML). Let us assume that graph G is partitioned into two crisp communities C_1 and C_2. To find the value of WFC, we consider the fraction of weighted connectivity over community structure C, i.e.

$$
\begin{aligned}
WFC(C) = & \frac{|V(C_1)|}{|V(C_1)|+|V(C_2)|}\frac{2|E(C_1)|}{|V(C_1)|(|V(C_1)|-1)} + \frac{|V(C_2)|}{|V(C_1)|+|V(C_2)|} \\
& \frac{2|E(C_2)|}{|V(C_2|(|V(C_2)|-1)} \\
& = \frac{2}{|V|}\{\frac{|E(C_1)|}{|V(C_1)|-1} + \frac{|E(C_2)|}{|V(C_2)|-1}\} \\
& = \frac{2}{|V|}\sum_{i=1}^{i=2}\frac{|E(C_i)|}{|V(C_i)|-1}
\end{aligned}
$$

We can generalize the WFC for community structure $C = \{C_1, C_2, C_3, \ldots, C_m\}$. The definitions of WFC and PML are given in Definition 4.5 and 4.6 respectively.

Definition 4.2.3. *(Weighted fully connectivity).* Given a graph $G(V,E)$ with non-overlapping community structure $C = \{C_1, C_2, C_3, \ldots, C_m\}$ then

we can estimate WFC as follows.

$$WFC(C) = \frac{2}{|V|} \sum_{i=1}^{i=m} \frac{|E(C_i)|}{|V(C_i)| - 1} \tag{4.5}$$

Definition 4.2.4. *(Percentage of missing links).* Given a graph $G(V,E)$ with crisp community structure $C = \{C_1, C_2, C_3, \ldots, C_m\}$ then we can estimate PMC as follows.

$$PML(C) = \{\frac{|E| - \sum_{i=1}^{i=m} |E(C_i)|}{|E|}\} * 100 \tag{4.6}$$

4.2.2.2 Finding non-desirable nodes

Before identifying seed nodes, we find non-desirable nodes using NDF algorithm. Non-desirable nodes are a set of nodes which are less likely to spread influence within the community. So these nodes are not the candidate for seed selection. The definition of non-desirable nodes is given in Definition 4.2.5.

Definition 4.2.5. *(Non-desirable nodes).* Non-desirable nodes are either hubs or nodes with large in-degree. These nodes have less influence spread within the community. However, we need seed nodes which are having more out-degree to same community.

As shown in FIGURE 4.2c, nodes $\{4,6,7,10\}$ are selected as non-desirable. Nodes $\{6,7\}$ are selected as hub nodes while $\{4,10\}$ are selected based on in-degree values.

4.2.2.3 Selecting seed nodes

Seed selection algorithm SSA works in two phases. In the first phase, $k_1 = f.k$ nodes are selected as seed with highest local diffusion degree values. In the second phase, $k_2 = (1-f).k$ nodes are selected with highest global diffusion degree values.

4.2.2.4 Computing spread of nodes

Influence spread in CIC model is similar to IC model. In IC model, every user u will influence their neighbors v with same influence probability of each topic. However, influence probabilities in CIC are different for each topic. Suppose user w activate user v through path $P = \{w = p_1, p_2, p_3, \ldots, p_l = v\}$ then we can estimate activation probability using Equation given below.

$$ap_{w,v} = \prod_{i=1}^{i=l} ap_{p_i,p_(i+1)} \tag{4.7}$$

To compute influence spread in CIC model, algorithm consider an user v which is activated by current seed set S with activation probability $ap(v|(S,Q))$, as follows.

$$ap(v|(S,Q)) = \begin{cases} 1 & v \in S \\ 1 - \prod_{w \in N_{inc}(v)} (1 - ap(w|(S,Q)).ap_{w,v}) & \text{otherwise} \end{cases} \tag{4.8}$$

Now, we can estimate the total influence of seed S by summing all the activation probabilities $ap(u|(S,Q)), u \in V$, i.e.

$$\sigma(S|Q) = \sum_{u \in V} ap(u|(S,Q)) \tag{4.9}$$

Algorithm 4: C2IM (G,Q): Proposed algorithm

Input: Directed graph $G(V,E)$, Contextual query Q, Number of Seed nodes k
Output: Seed set S

$S \leftarrow \phi$
$G_C \leftarrow G$
$f \leftarrow 0.9$
$C \leftarrow$ CDA(G) ▷ See **Algorithm 5**
$Adj \leftarrow$ Adjacency list for graph $G(V,E)$
$Rev \leftarrow$ Adjacency list with reversed edges for graph $G(V,E)$
Sort adjacency list of *Rev* and *Adj* in decreasing order of activation probability
for *each community* $C_i \in \{C_1, C_2, \ldots, C_m\}$ **do**
    $k_i \leftarrow \frac{|C_i| * f * k}{|V|}$
    $S_i \leftarrow \text{SSA}(C_i, k_i)$ ▷ See **Algorithm 7**
    $S \leftarrow S_i \cup S$
$S_{G_C} \leftarrow \text{SSA}(G_C, k - |S|)$ ▷ See **Algorithm 7**
$S \leftarrow S_{G_C} \cup S$
Return S

Algorithm 5: CDA (G): Community Detection Algorithm

Input: Social Graph $G(V,E)$
Output: Community Structure $C = \{C_1, C_2, C_3, \ldots, C_m\}$

```
1  C ← φ
2  C_Temp ← {V_1, V_2, ..., V_N}
3  for i = 1 to |V| − 1 do
4      C_j ← Select a random community from C
5      density ← 0
6      CMerge ← φ
7      for each community C_i ∈ C_Temp do
8          if C_i ≠ C_j then
9              temp ← Δ_{C_i,C_j}
10             if temp > density then
11                 density = temp
12                 CMerge ← C_i
13                 else if temp = density and CMerge ≠ φ then
14                     if GetCommunity(C_j, C_i) > GetCommunity(C_j, CMerge) then
15                         CMerge = C_i
16     if CMerge ≠ φ then
17         $CMerge ← Merge (CMerge, C_j)
18     else C ← C ∪ C_j
19     ;
20     C_Temp ← C_Temp \ C_j
21 Return C = {C_1, C_2, C_3, ..., C_m}
```

4.3 Algorithm

The main **Algorithm 4** takes three inputs, a directed graph G, topic-based contextual query Q and the number of seed nodes k. Lines 1-3 initialize S, G_C, and f with the empty set, G and 0.9 respectively. Line 4 finds the community structure of influence graph G. Lines 5-6 maintain adjacency lists Adj and Rev of G with exact and reverse edge direction respectively. Line 7 sorts both adjacency lists Adj and Rev in decreasing order based on their activation probability. The **for** loop in lines 8-11, scan each

Algorithm 6: NDF (C_i, k_i): Non-Desirable nodes Finder

Input: Community C_i and number of non-desirable nodes k_i
Output: Set of non-desirable nodes ND_{C_i}

```
1  h ← 0.8
2  Hub ← h * k_i
3  ND_{C_i} ← φ
4  for each node u ∈ C_i do
5      insider[u] ← 0
6      outsider[u] ← 0
7  for each node u ∈ C_i do
8      for each edge (u,v) ∈ Adj[u] do
9          if v ∈ C_i then
10             insider[v] ← insider[v] + 1
11         else outsider[u] ← outsider[u] + 1
12         ;
13 for each node u ∈ C_i do
14     temp_u ← outsider[u] * insider[u]
15 ND_{C_i} ← ND_{C_i} ∪ Hub number of nodes with highest temp_u values
16 temp ← k − |ND_{C_i}|
17 ND_{C_i} ← ND_{C_i} ∪ temp number of nodes with highest |Adj[u]| values
18 Return ND_{C_i}
```

community and identify seed in communities. Line 9 estimates required seed size k_i for each community C_i. Line 10 finds seed nodes S_i for each community. Line 11 adds community seed S_i to seed set S. Line 12 finds seed S_{G_C} considering graph G as a single community. Line 13 adds $(k - |S|)$ seed S_{G_C} to seed set S. **Algorithm 4** returns seed set S for graph G (line 14).

Algorithm 5 takes graph G as input. Lines 1-2 initialize C and C_{Temp} with the empty set and set of nodes $V_i \in V$. Initially, each node is considered as an individual community. The **for** loop in lines 3-19, find community structure C of graph G. Line 4 selects a random community C_j. Lines 5-6

Algorithm 7: SSA (C_i, k_i): Seed Selection Algorithm

Input: Community C_i and number of Seed nodes k_i
Output: Seed Set S_i

```
1  r ← 0.2
2  S_i ← φ
3  NonDec =NDF(C_i, r * |C_i|)                    ▷ See Algorithm 6
4  for i = 1 to |C_i| do
5      d[i] ← 0
6      vis[i] ← 0
7  Q_ND ← NonDec
8  while Q_ND ≠ φ do
9      Q_new ← φ
10     while Q_ND ≠ φ do
11         v ← Q_ND.front()
12         Q_ND.pop()
13         for each u ∈ Rev[v] and u ∈ C_i do
14             if vis[u] = 0 then
15                 vis[u] ← 1
16                 Q_new.push(u)
17             d[u] ← (d[u] + d[v] + ap_{u,v})
18     Q_ND ← Q_new
19 Sort nodes u ∈ C_i in decreasing order of d[u] values
20 for each node u ∈ C_i in decreasing order of d[u] do
21     if u.flag ≠ 1 then
22         u.flag ← 1
23         S_i ← u
24         for each node v ∈ N_out(u) do
25             v.flag ← 1
26         if |S_i| = k then
27             return S_i
28 return S_i
```

initialize *density* and *CMerge* with zero and empty set respectively. The **for** loop in lines 7-15, finds a community C_i which produces the highest density after merging with C_j. Line 8 checks that communities C_i and C_j are equivalent or not. Line 9 estimates change in density after merge communities C_i and C_j. Lines 10-15 identify the community C_i which is

best possible pair with C_j and store it in $CMerge$. Lines 16-18 merge the community C_j with $CMerge$ iff $CMerge$ is not empty, otherwise add C_j to community structure C. Line 19 removes C_j from C_{Temp}. **Algorithm 5** returns community structure C of graph G (line 20).

Algorithm 6 takes two inputs, community C_i and seed size k_i. Lines 1-3 initialize parameter h, Hub and ND_{C_i} with 0.8, $h * k_i$ and empty set respectively. The **for** loop in lines 4-6 repeatedly initializes $insider[u]$ and $outsider[u]$ for each node $u \in C_i$ with zero. The **for** loop in lines 7-11 repeatedly updates the values of $insider[u]$ and $outsider[u]$ based on their intra-connections and inter-connections respectively. The **for** loop in lines 12-13 repeatedly finds the maximum number of possible ways to spread from u. Line 14 finds the non-desirable nodes based on the number of possible ways to spread. Line 15 finds the number of nodes which we want to select as non-desirable nodes based on $|Adj[u]|$ values. Line 16 finds the non-desirable nodes based on minimum influence spread nodes. **Algorithm 6** returns the set of non-desirable nodes ND_{C_i} of C_i (line 17).

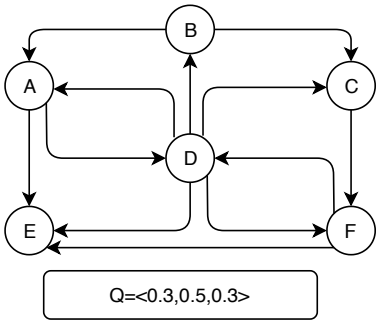
Algorithm 7 takes two inputs, community C_i and seed size k_i. Line 1 initializes the fraction of nodes r to be non-desirable nodes in the community C_i with 0.2. Line 2 initializes the seed set S_i with the empty set. Line 3 extracts r fraction of non-desirable nodes in the community C_i. The **for** loop in lines 4-6, initializes the two list d and vis of size $|C_i|$ with zero. Line 7 initializes non-desirable queue Q_{ND} with the set of non-desirable nodes $NonDec$ of community C_i. The **while** loop in lines

8-18 repeatedly calculates the diffusion degree $d[u]$ of each node in the community. Line 8 checks, Q_{ND} is empty or not. Line 9 initializes queue Q_{new} with an empty set. The **while** loop in lines 10-17 repeatedly updates $d[u]$ values until Q_{ND} is non-empty. Line 11 extracts front node of queue Q_{ND} and place into queue Q_{new}. Line 12 pops out the node from the front of the queue Q_{ND}. The **for** loop in lines 13-17 iteratively traverses community in reverse direction from non-desirable nodes Q_{ND}, and update $d[u]$ values. Lines 14-16 check, if $vis[u] = 0$ then set $vis[u]$ to 1 and add u to the rear of queue Q_{new}. Line 17 updates the value of diffusion degree $d[u]$. Line 18 initializes queue Q_{ND} with Q_{new}. Line 19 performs sorting of nodes based on their $d[u]$ values. The **for** loop in lines 20-27, iteratively selects seed nodes and avoid overlapping spread to maximize influence spread. Lines 21-27 check, if node u is not selected as seed by now then selects u as seed node. Also, mark u and its neighbors to be selected, to reduce overlapping influence. **Algorithm 7** outputs the seed set S_i for community C_i (Line 28).

4.3.1 Applying the Algorithm

The running example shows in FIGURE 4.3. We consider given graph as a single community. Topic distribution in running example is also given. FIGURE 4.3 shows the colored nodes E and C which are selected as non-desirable nodes based on NDF. After selecting non-desirable nodes, first, we need to reverse the edges of our graph as shown in FIGURE 4.3.

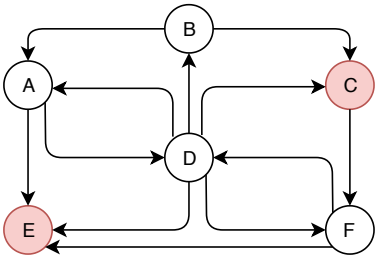
Example Graph



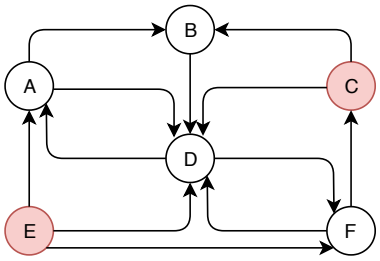
Topic Distribution

Edge	A–D	A–E	B–A	B–C	C–F	D–A
T_1	0.4	0.6	0.2	0.4	0.1	0.2
T_2	0.3	0.1	0.3	0.2	0.4	0.3
T_3	0.2	0.0	0.0	0.6	0.2	0.0
Edge	D–B	D–C	D–E	D–F	F–D	F–E
T_1	0.3	0.3	0.4	0.2	0.4	0.2
T_2	0.4	0.4	0.5	0.1	0.2	0.1
T_3	0.5	0.5	0.0	0.6	0.1	0.3

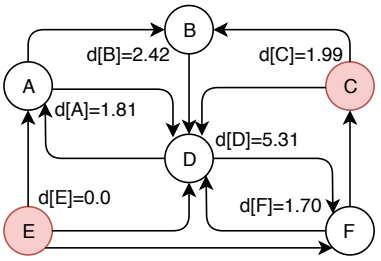
Non-desirable Nodes



Reverse Edge Graph



Diffusion Degree $d[u]$



Seed Nodes

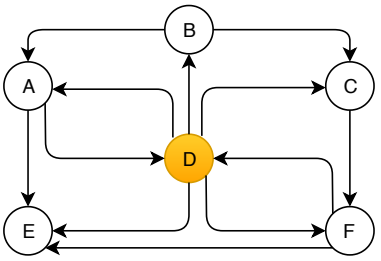


FIGURE 4.3: A Running Example

Now, we find the local influence spread from non-desirable nodes in reverse direction as shown in TABLE 4.1. Then we find out the final influence spread on each node of the community.

TABLE 4.1: Estimation of Diffusion Degree based on SSA in Running Example

Q_{ND}	Nodes $< vis[], d[] >$						Q_{new}	v
	A	B	C	D	E	F		
CE	0,0	0,0	0,0	0,0	0,0	0,0	ϕ	–
E	0,0	1,0.40	0,0	1,0.44	0,0	0,0	BD	C
ϕ	1,0.23	1,0.40	0,0	1,0.81	0,0	1,0.20	$BDAF$	E
DAF	1,0.23	1,0.40	0,0	1,1.25	0,0	1,0.20	ϕ	B
AF	1,1.81	1,0.40	0,0	1,1.25	0,0	1,1.70	ϕ	D
F	1,1.81	1,2.42	0,0	1,3.32	0,0	1,1.70	ϕ	A
ϕ	1,1.81	1,2.42	1,1.99	1,5.31	0,0	1,1.70	ϕ	F

For example, spread starts from the non-desirable nodes C and E based on *SSA*. Therefore, we add C and E node to Q_{ND} and pop front node of Q_{ND}

and store it to v. FIGURE 4.3 shows that node C spread influence to its out-neighbors B and D. Nodes B and D are placed into Q_{new}. After that, we update $vis[B]$ and $vis[B]$ with 1. After that the value of diffusion degree of B and D is estimated using equation $d[u] \leftarrow (d[u]+d[v]+ap_{u,v})$. Diffusion degree of nodes B and D are estimated as $d[B] \leftarrow (0+0+(0.4*0.3+0.2*0.5+0.6*0.3))$ and $d[D] \leftarrow (0+0+(0.3*0.3+0.4*0.5+0.5*0.3))$ respectively. Therefore, $d[B]$ and $d[D]$ is set to 0.40 and 0.44 respectively. Similarly, we can estimate diffusion degree for whole community based on *SSA* as shown in TABLE 4.1. FIGURE 4.3 shows the final diffusion degree of each node. Highest diffusion degree node D is selected as seed node.

4.3.2 Complexity Analysis

Our proposed algorithm C2IM performs initialization in lines 1-3, so it takes constant time. In line 4, it computes community structure C using CDA algorithm and takes $O(|V|^2Z)$ time where Z is the average number of edges in communities. Lines 5-6 take $O(|E|)$ time. Line 7 performs sorting on E, so it takes $O(|E|\log|E|)$ time. The **for** loop in lines 8-11, finds seed set for each community using SSA algorithm and take $O((|V|+|E|)\log(|V|+|E|))$ time. Line 13 takes $O((|V|+|E|)\log(|V|+|E|))$ time. Hence, C2IM performs in $O((|V|+|E|)\log(|V|+|E|))$ time when community structure is known, otherwise $O(|V|^2Z)$. Theoretical complexity of our algorithm is not much improved, as it still computes $(1-f)$ fraction of seed nodes based on

global influence spread. While practically our algorithm is more efficient because most of seed nodes $(f\%)$ are selected based on local influence spread. The proposed algorithm is super-linear when community structure is known.

4.4 Empirical Analysis

In this section, firstly, the experimental setup information is provided along with the dataset, seeding strategies, and probability distribution for diffusion models. Further, discussion of the performance analysis regarding influence spread and efficiency along with parameter analysis are provided. Finally, illustrate the statistical test to show the significant difference between the proposed method and the compared seeding methods.

4.4.1 Experimental Setup

All the experiments performed on six real-world network datasets: BHOSLIB [140], Dolphin [138], Gnutella08 [143], Astro-ph [140], AS-22JULY06 [140], and Gnutella30 [143] regarding influence maximization. The performance of community detection algorithm CDA tested on Email-Eu-core [141] dataset. The performance of proposed algorithm is tested against four seeding strategies: Random [11],

MaxDegree [11], CIM [49], and TIM [21]. C2IM-WC is another version of proposed algorithm which does not require community information, instead of that it considers whole network as a single community.

The algorithm utilizes well-known probability distributions to assign influence probabilities. The propagation probability for each edge follows a uniform distribution. The propagation probability in IC models assigned based on uniform distribution over $\{0.1,0.01,0.001\}$. For the LT model, propagation probability $p_{x,y}$ is assigned by $\frac{1}{indegree(y)}$. The activation probability of a node x follows uniform distribution over [0,1]. The *adjacency list* is used for storing graph G and *Red-Black tree* (*C++ std :: set*) is used for merging and storing communities. A user-defined data structure *cluster* is used to store community informations such as, the nodes in community, edges within the community and the edges from a node in community to node in some other communities, etc. *Vector* is used for other purposes. Each of the methods is implemented using C++ language and executed on DEV-C++. All of the algorithms run 20 times independently on a 64-bit window's PC with Intel(R) Core(TM) i7-7700 CPU@ 3.60GHz processor and 8GB memory.

4.4.2 Parameter Setting

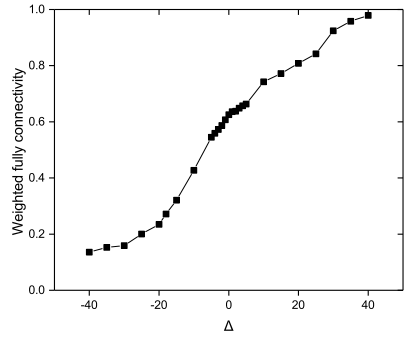
Three parameters f, r, and h are used in our paper. Parameter $f \in [0,1]$ is the fraction of seed to be taken from communities. f fraction of seed are

selected based on their local influence. Remaining $(1-f)$ fraction of seed are selected based on their global influence. Parameter $r \in [0,1]$ is the fraction of nodes in the community to be selected as non-desirable. Parameter $h \in [0,1]$ is the fraction of nodes which are selected as non-desirable from each community. h fraction of nodes are selected based on their inter-connection. Remaining $(1-h)$ are taken from maximum in-degree nodes. We use parameter values $f = 0.8$, $r = 0.2$, and $h = 0.8$ in our experiment.

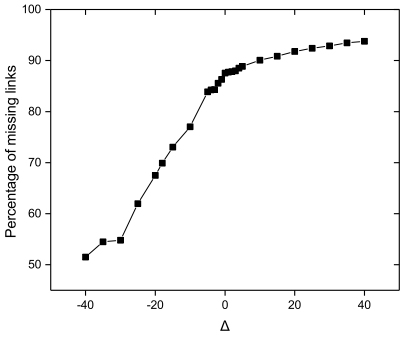
4.4.3 Varying Δ

In general, community detection process is considered as a preprocessing step. We experiment in dataset *email-Eu-core* [141] to analyze the performance of community detection algorithm CDA with different Δ[1] from -40 to 40. We do experiment on Δ to merge the generated communities in previous iteration and show the effect of threshold value to control the size of community structure. When $\Delta = -40$, almost each communities are merged and number of communities is only 12; when $\Delta = 40$, the generated communities are very small and dispersed (expected community size is 1.88). We can see in Figure 4.4, if the value of parameter Δ is small then, big size densely connected communities are generated. Otherwise small size sparsely connected communities are generated. Therefore, algorithm need to choose a threshold value which

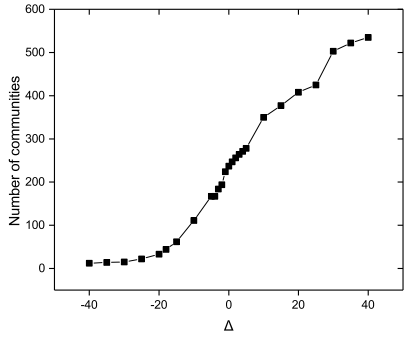
[1] $\{D(C_{i,j}) - (D(C_i) + D(C_j)) \geq \Delta_{C_i,C_j} * D(C_{i,j})\} -> \Delta$



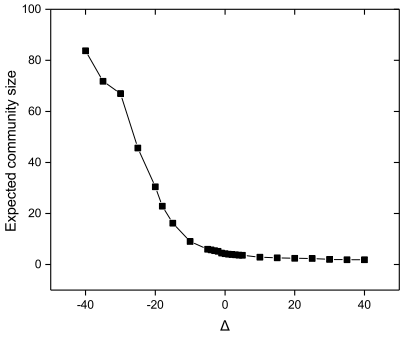
(A) Distribution of WFC



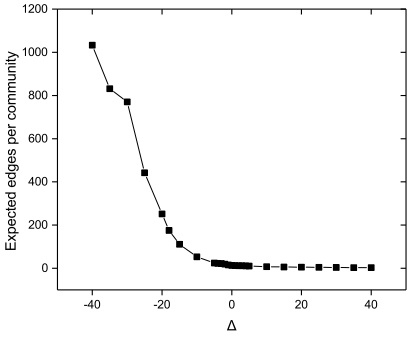
(B) Distribution of PML



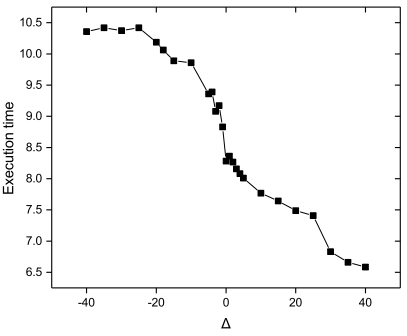
(C) Distribution of number of community



(D) Distribution of community size



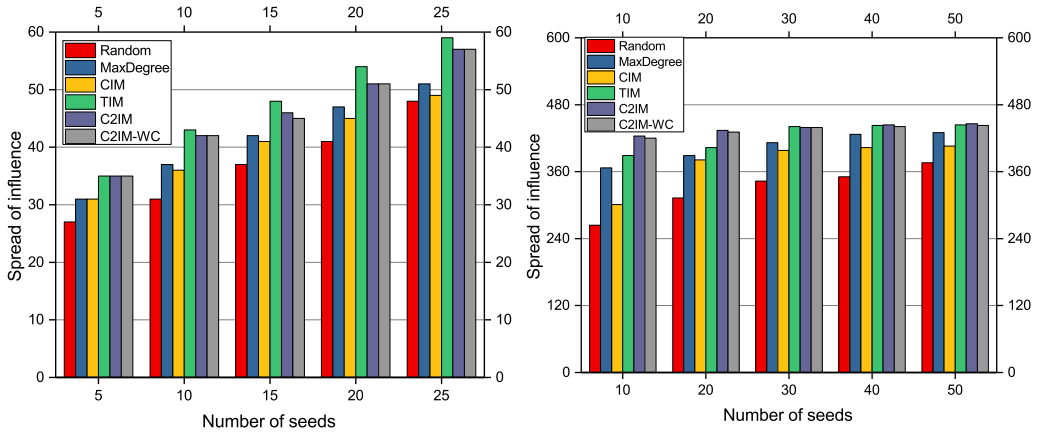
(E) Distribution of edges per community



(F) Distribution of execution time (in sec.)

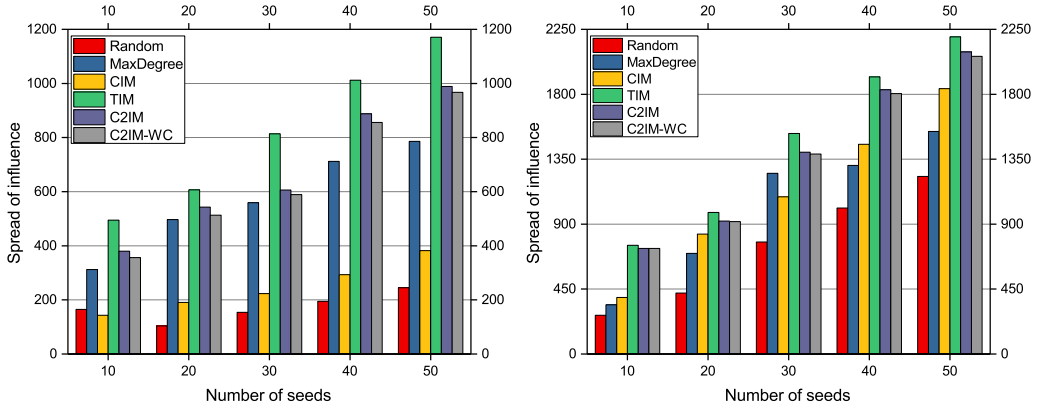
FIGURE 4.4: Distribution of Various Measures over Δ in Email-Eu-core Dataset

generate medium-size communities with more intra-connection (densely connected) to maximize local influence.



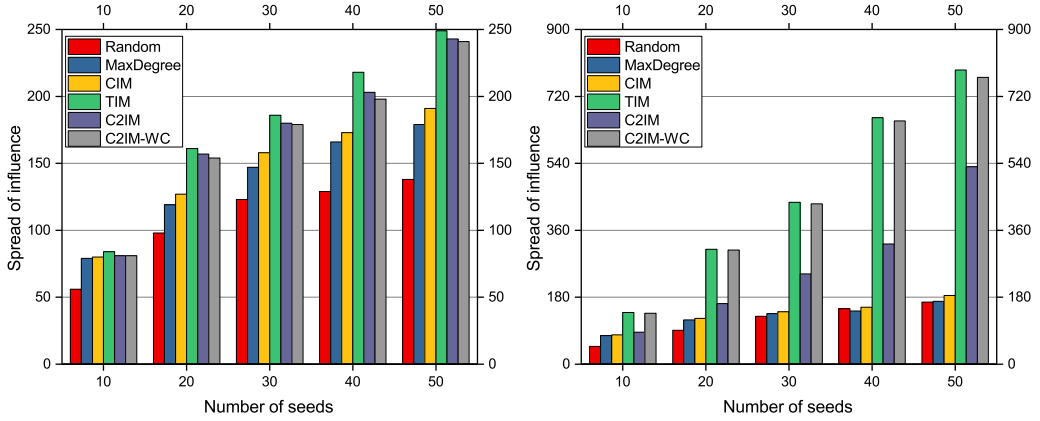
(A) Dolphin

(B) BHOSLIB



(C) Gnutella08

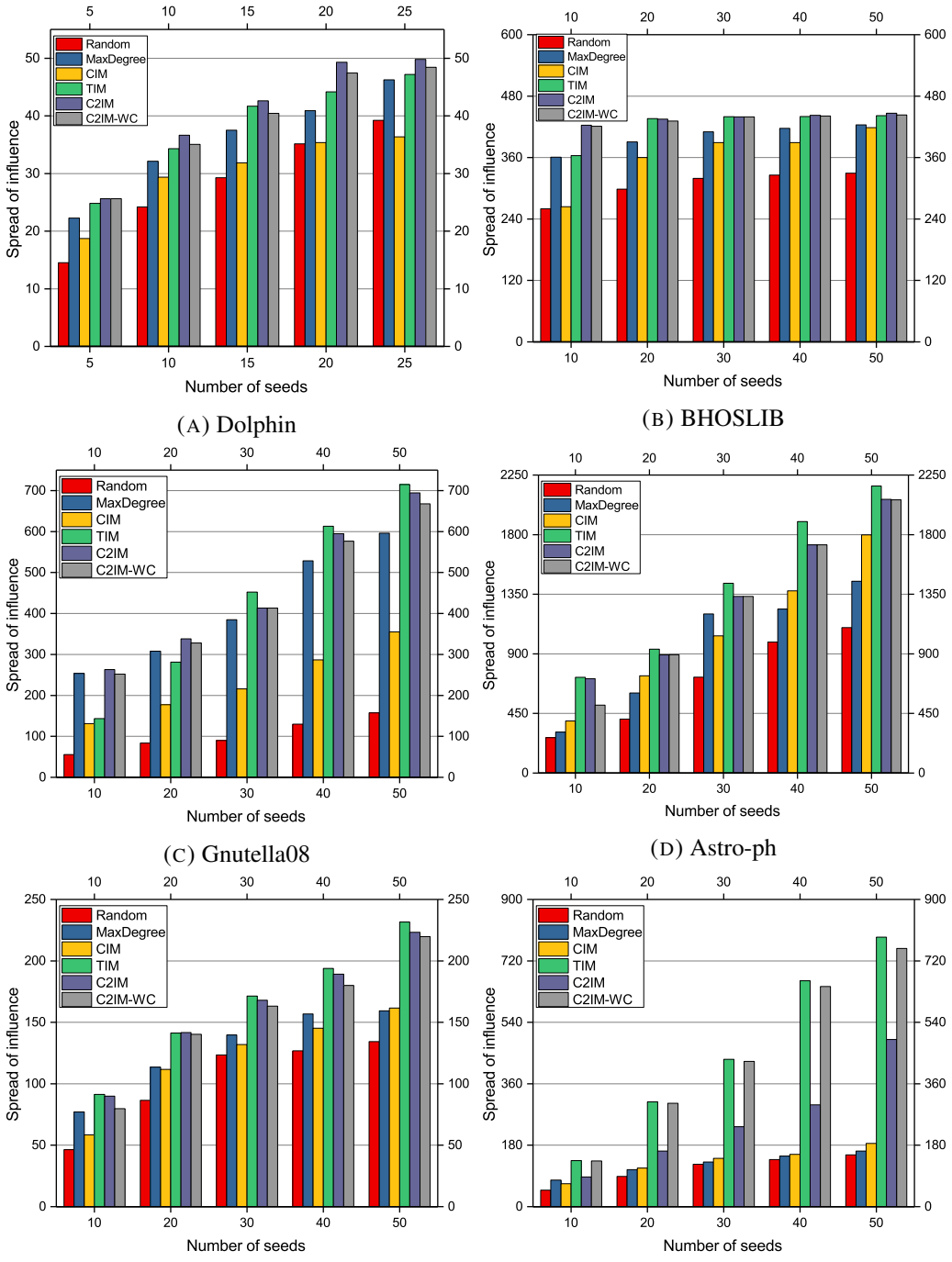
(D) Astro-ph



(E) As-22july06

(F) Gnutella30

FIGURE 4.5: Influence Spread Comparison in Various Datasets under LT Model



(A) Dolphin (B) BHOSLIB

(C) Gnutella08 (D) Astro-ph

(E) As-22july06 (F) Gnutella30

FIGURE 4.6: Influence Spread Comparison in Various Datasets under IC Model

4.4.4 Analyzing Quality

Quality in influence maximization problem equates to the expected influence spread from seed users S. An algorithm with higher influence

TABLE 4.2: Speedup % of Proposed Algorithms against the State-of-the-art Algorithms in Different Datasets regarding Influence Spread

Model	Algorithm Compared	Seed set size	Datasets									
			BHOSLIB		Gnutella08		Astro-ph		AS-22JULY06		Gnutella30	
			C2IM	C2IM-WC	C2IM	C2IM-WC	C2IM	C2IM-WC	C2IM	C2IM-WC	C2IM	C2IM-WC
LT	Random	10	60.61	59.09	130.30	115.76	172.12	172.12	44.64	44.64	79.17	185.42
		20	38.66	37.70	422.12	393.27	117.73	116.79	60.20	57.14	79.12	237.36
		30	27.99	27.99	293.51	282.47	80.28	78.61	46.34	45.53	88.37	234.11
		40	26.50	25.64	355.39	338.97	80.93	78.36	57.36	53.49	116.78	338.93
		50	18.62	17.82	303.67	294.69	70.02	67.59	76.09	74.64	217.96	361.68
	MaxDegree	10	15.53	14.44	21.80	14.10	114.66	114.66	2.53	2.53	11.69	77.92
		20	11.57	10.80	9.26	3.22	31.95	31.38	31.93	29.41	36.98	157.98
		30	6.55	6.55	8.41	5.37	11.74	10.70	22.45	21.77	78.68	216.91
		40	3.98	3.28	24.72	20.22	40.09	38.10	22.29	19.28	125.87	357.34
		50	3.72	3.02	25.83	23.03	35.73	33.79	35.75	34.64	214.20	356.21
	CIM	10	40.86	39.53	165.73	148.95	86.73	86.73	1.25	1.25	8.86	73.42
		20	13.91	13.12	184.29	168.59	10.83	10.35	23.62	21.26	32.52	149.59
		30	10.30	10.30	171.75	164.13	28.47	27.27	13.92	13.29	72.34	205.67
		40	10.17	9.43	203.07	192.15	26.02	24.23	17.34	14.45	111.11	327.45
		50	9.85	9.11	158.90	153.14	13.81	12.18	27.23	26.18	187.03	316.76
	TIM	10	9.00	7.97	-23.23	-28.08	-2.92	-2.92	-3.57	-3.57	-38.13	-1.44
		20	7.69	6.95	-10.54	-15.49	-6.12	-6.52	-2.48	-4.35	-47.25	-0.65
		30	-0.45	-0.45	-25.55	-27.64	-8.50	-9.35	-3.23	-3.76	-44.14	-0.92
		40	0.23	-0.45	-12.25	-15.42	-4.69	-6.04	-6.88	-9.17	-51.28	-1.36
		50	0.45	-0.23	-15.54	-17.42	-4.78	-6.14	-3.95	-4.74	-32.87	-2.53
IC	Random	10	62.80	62.05	372.10	352.52	166.63	91.71	94.02	72.24	79.99	177.27
		20	45.95	44.60	304.10	292.16	119.51	119.70	63.63	61.94	83.57	241.24
		30	37.65	37.64	358.42	358.29	84.28	84.28	36.06	32.15	88.83	243.03
		40	35.85	35.50	357.83	343.95	74.21	74.16	49.22	41.94	116.17	367.24
		50	35.53	34.52	341.18	324.30	88.18	87.80	66.18	63.47	222.69	398.68
	MaxDegree	10	17.28	16.74	3.70	-0.60	130.45	65.70	16.71	3.61	11.84	72.28
		20	11.46	10.43	9.66	6.42	47.65	47.78	24.71	23.42	50.48	179.72
		30	7.13	7.12	7.35	7.31	11.09	11.09	20.29	16.83	78.78	224.77
		40	6.06	5.79	12.48	9.07	39.14	39.10	20.67	14.78	101.68	335.94
		50	5.38	4.59	16.46	12.00	42.77	42.48	40.09	37.80	200.16	363.86
	CIM	10	60.38	59.64	100.78	92.45	81.62	30.59	53.89	36.61	29.87	100.07
		20	20.89	19.77	90.78	85.14	21.48	21.58	26.86	25.56	44.23	168.11
		30	12.92	12.91	91.50	91.45	28.77	28.77	27.40	23.74	65.83	201.25
		40	13.69	13.40	107.40	101.11	25.25	25.22	30.27	23.91	94.57	320.56
		50	6.77	5.97	95.63	88.14	14.96	14.73	38.15	35.89	163.85	307.75
	TIM	10	16.31	15.77	83.62	76.00	-1.42	-29.12	-1.57	-12.62	-35.57	-0.74
		20	-0.16	-1.08	20.12	16.57	-4.54	-4.45	0.26	-0.77	-46.91	-1.30
		30	-0.09	-0.09	-8.61	-8.63	-6.89	-6.89	-1.93	-4.75	-45.72	-1.39
		40	0.51	0.25	-3.02	-5.96	-9.22	-9.25	-2.37	-7.13	-54.92	-2.57
		50	1.11	0.36	-2.85	-6.56	-4.64	-4.83	-3.55	-5.12	-38.00	-4.18

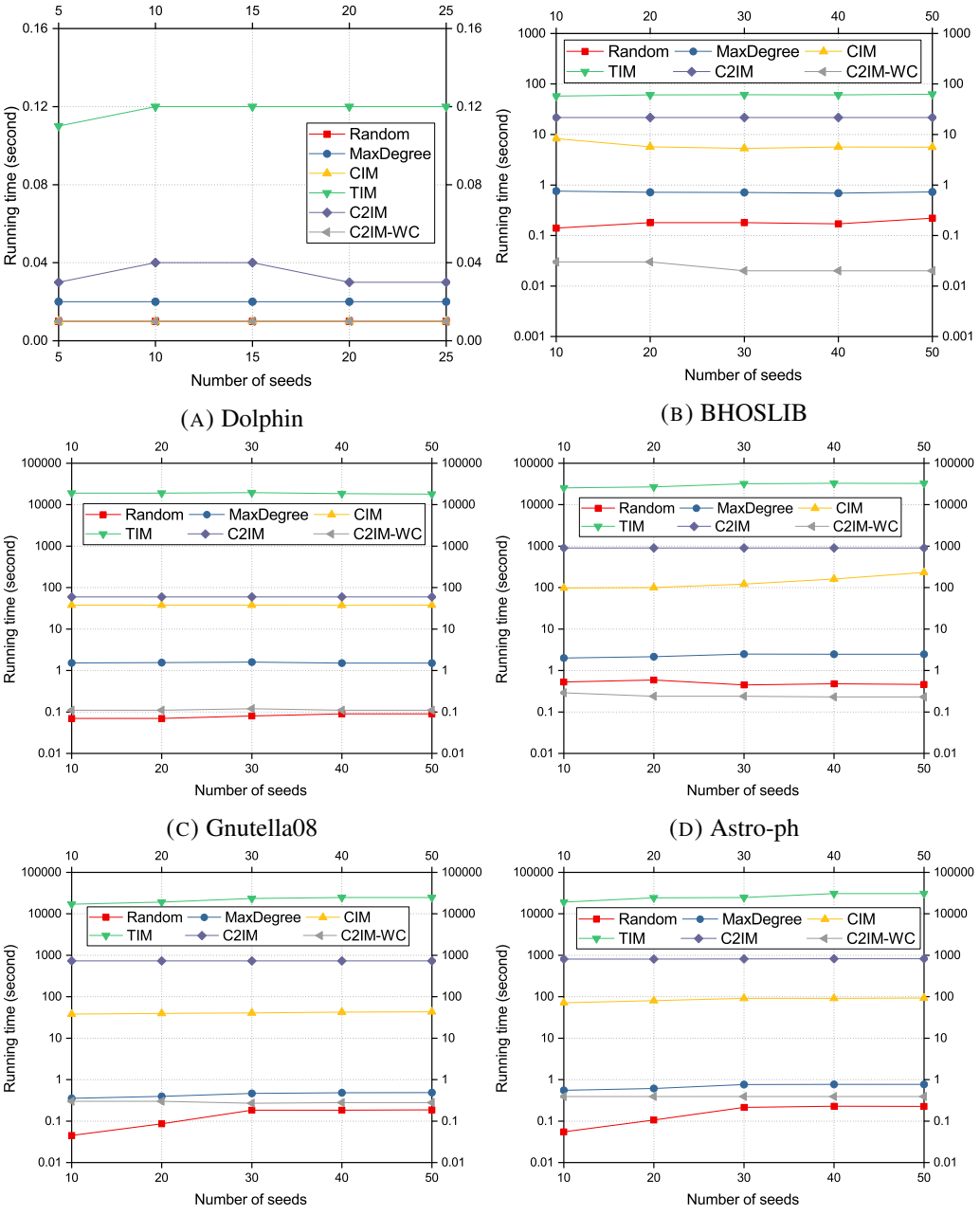
spread in the network is termed as higher quality algorithm. For quality analysis of proposed algorithm, we select 5 different set of seed nodes of size 10, 20, 30, 40 and 50 for *BHOSLIB*, *Gnutella*08, *Astro-ph*, *As*-22*july*06 and *Gnutella*30 datasets. To quality analysis in small-scale (*Dolphin*) dataset we select 5 different set of seed nodes of size 5, 10, 15, 20 and 25. We use CIM and TIM as base algorithms for our work.

TABLE 4.3: Speedup % of Proposed Algorithms against the State-of-the-art Algorithms in Different Datasets regarding Running Time

Algorithm Compared	Seed set size	Datasets									
		BHOSLIB		Gnutella08		Astro-ph		AS-22JULY06		Gnutella30	
		LT	IC	LT	IC	LT	IC	LT	IC	LT	IC
CIM	10	-161.55	-130.22	-57.11	-58.42	-825.95	-890.64	-818.81	-948.64	-835.60	-996.82
	20	-281.87	-254.70	-58.20	-59.36	-801.76	-863.70	-751.69	-852.86	-908.39	-907.81
	30	-308.75	-275.73	-57.34	-58.61	-640.40	-692.08	-704.86	-658.21	-791.59	-802.37
	40	-284.49	-256.80	-59.68	-60.65	-457.69	-496.64	-616.50	-555.08	-799.85	-795.75
	50	-288.68	-260.10	-57.70	-58.93	-287.86	-314.92	-588.11	-530.03	-783.52	-771.69
TIM	10	62.19	61.40	101.69	103.65	96.45	96.35	95.70	95.60	95.76	95.67
	20	64.21	65.97	99.69	101.65	103.63	96.53	96.18	96.17	96.67	96.70
	30	64.44	62.99	99.69	101.69	103.16	97.23	96.87	97.08	96.70	96.86
	40	64.27	60.99	99.68	101.66	103.25	97.38	97.05	97.22	97.33	97.50
	50	65.40	63.94	99.67	101.65	103.23	97.44	97.05	97.78	97.33	97.63

FIGURE 4.5 and 4.6 show that the comparison of influence spread of our proposed algorithm with the state-of-the-art algorithms under LT and IC diffusion model respectively. FIGURE 4.5 and 4.6 show that our proposed algorithms C2IM and C2IM-WC outperform the heuristic approaches Random and MaxDegree in terms of influence spread under both diffusion models. Our proposed algorithms outperform the CIM and quite close to TIM in quality under LT and IC models.

TABLE 4.2 shows the speedup % (in terms of influence spread) of our proposed algorithms C2IM and C2IM-WC over baseline algorithms. We can see that our proposed algorithms have positive speedup over baseline approaches except TIM algorithm. To acquire trade-off between quality and efficiency, our algorithms compromise little bit in quality to gain high speedup in terms of running time over TIM.



(A) Dolphin (B) BHOSLIB

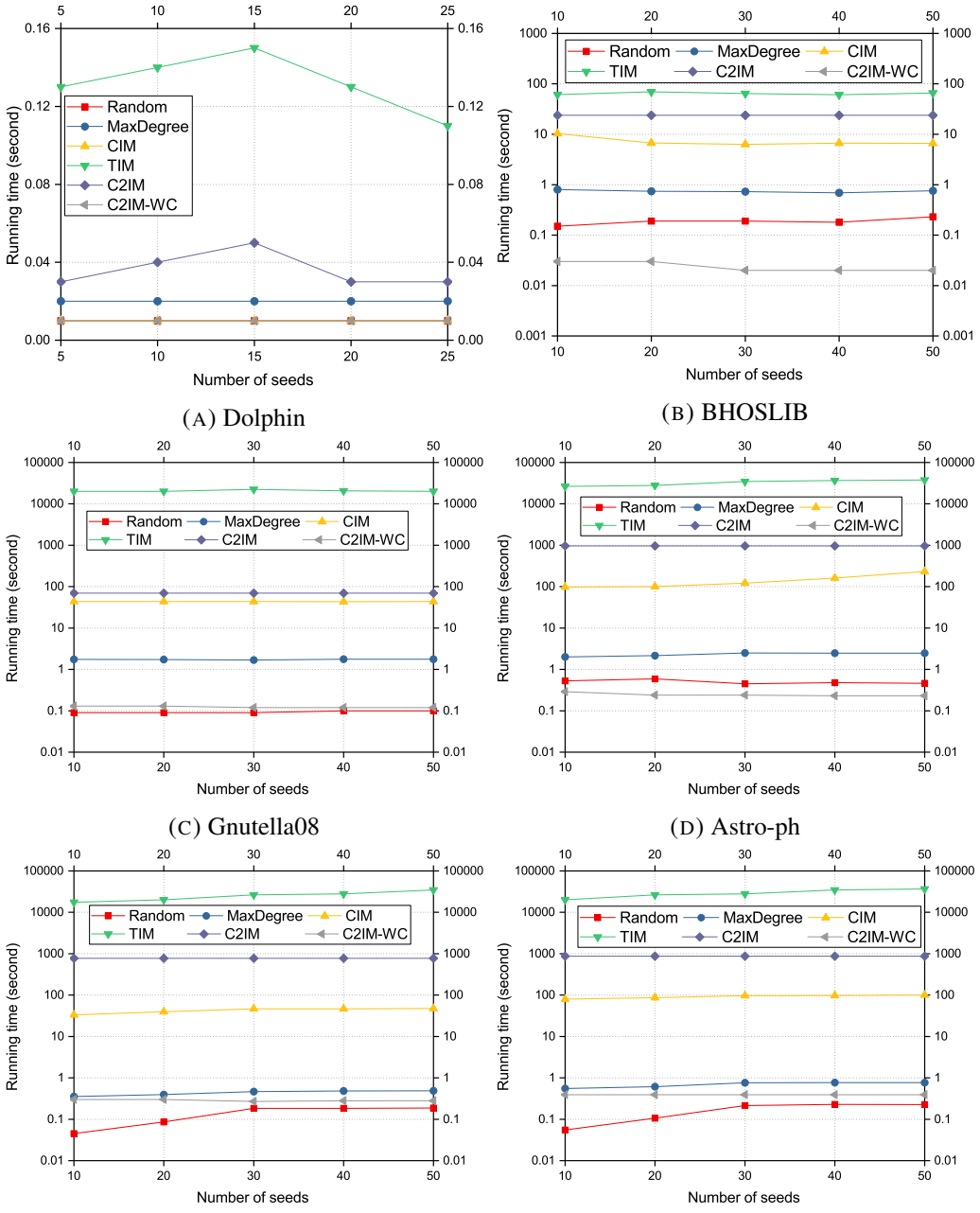
(C) Gnutella08 (D) Astro-ph

(E) As-22july06 (F) Gnutella30

FIGURE 4.7: The Running Time Comparison in Various Datasets under LT Model

4.4.5 Analyzing Efficiency

To study how efficiently proposed algorithm C2IM works on different size seed set, we compare running time of C2IM with the state-of-the-art



(A) Dolphin (B) BHOSLIB

(C) Gnutella08 (D) Astro-ph

(E) As-22july06 (F) Gnutella30

FIGURE 4.8: The Running Time Comparison in Various Datasets under IC Model

algorithms. In order to efficiency analysis of proposed algorithm, we experiment on same size set of seed nodes as we use in quality analysis on six real-world datasets.

FIGURE 4.7 and 4.8 show the comparison of running time of proposed algorithms with the state-of-the-art algorithms under LT and IC diffusion model respectively. Heuristic approaches Random and MaxDegree take almost same running time except additional sorting time based on the degree of nodes in MaxDegree algorithm. Both heuristic approaches do not consider any feature, context or information in diffusion process. Thus both methods are very efficient but generate poor quality seed set. Community-based influence maximization CIM reduces the search space for seed selection so CIM generates seed set efficiently. Seed quality of CIM is poor because seed selection is based on local influence and it does not consider the global influence of node as well as user interest. TIM generates good quality seed however it uses time consuming maximum influence arborescence (MIA) structure. Therefore, the running time of TIM is large. In FIGURE 4.7 and 4.8, we can see that our algorithm maintains a trade-off between CIM and TIM.

TABLE 4.3 shows the speedup % (in terms of running time) of proposed algorithm C2IM over the base algorithms CIM and TIM under LT and IC models. We can see that our proposed algorithms have positive speedup over TIM and negative speedup over CIM. CIM works very efficiently in very sparse datasets (As-22july06 and Gnutella30) because of both the dataset generate very small community which reduces the search space significantly. Thus the quality of seed in CIM also degrades significantly. TABLE 4.2 and 4.3 show that our proposed algorithm C2IM finds good

quality effective seed efficiently.

4.5 Conclusion

In this chapter, the role of contextual feature and community-structure is analyzed and a community-based context-aware influence maximization algorithm is presented. The algorithm focused on two issues in traditional IM problem, first time-efficiency and second effectiveness of seed. C2IM uses the community-based framework to improve the efficiency and user's interest to select effective and accurate seed users. The analysis revealed following facts.

- The expected influence spread with contextual query Q is submodular under extended diffusion models CIC and CLT. Also, C2IM problem under these diffusion models is NP-hard.
- The algorithm utilizes community structure to find most influential users efficiently like CIM [49]. Though, C2IM finds influential users based on nodes local influence as well as global influence unlike CIM.
- C2IM considers contextual features, eg., user's interest and product characteristics for IM. The social influence between two users depends not only relationship between them but also the contextual features unlike CIM.

- The algorithm identifies non-desirable nodes as non-candidate of influential users for backward influence propagation unlike CIM and TIM [21]. Also, devise new method to estimate diffusion degree of each node locally and globally using backward propagation unlike CIM and TIM.

- The empirical results show that the proposed algorithm performs up to 3.27 and 3.20 times better than CIM w.r.t.[2] influence spread under CLT and CIC model respectively while comparable w.r.t. running time. C2IM performed up to 1.01 and 1.03 times faster than TIM w.r.t. running time under LT and IC model respectively while comparable w.r.t. influence spread.

- In conclusion, C2IM algorithm obtains a good trade-off between effectiveness and efficiency.

[2]with respect to