Minimizing Dissemination in a Population While Maintaining its Community Structure

Chuxu Zhang
Department of Computer Science
Rutgers University
cz201@cs.rutgers.edu

Tina Eliassi-Rad
Department of Computer Science
Rutgers University
eliassi@cs.rutgers.edu

ABSTRACT

Given a population represented as a graph (e.g., a social network where the nodes are individuals and the links indicate friendships), can we minimize the spread of an entity (such as a meme or a virus) while maintaining the population's community structure (i.e., dense connections among nodes within a community but sparse connections between nodes in different communities)? At first glance, these two objectives seem at odds with each other. To minimize dissemination, nodes or links in the graph are often deleted to reduce the population's connectivity. These deletions can (and often do) destroy the community structure present in the population, which is an important construction in society. We introduce the strategy of *rewiring* links to achieve both objectives. Examples of rewiring in real life are prevalent, such as purchasing products from a new farm since the local farm has signs of mad-cow disease; getting information from a new source after a disaster since your usual source is not longer available, etc. This paper has three parts. First, we formally introduce the problem of minimizing dissemination on a population (represented as a graph) while maintaining its community structure by rewiring a set of links. Second, we propose two effective and efficient algorithms: *CRlink* (short for Community Relink) and *constrCRlink* (short for Constraint Community Relink). Third, we present the results of extensive experiments on different graphs to show that our algorithms perform well in both minimizing dissemination and maintaining community structure. When compared with the most effective algorithm for minimizing spread on a graph through link deletions (namely, *NetMelt*$^+$), *constrCRlink* preserves (on average) 98.6% of *NetMelt*$^+$'s efficacy in minimizing dissemination and only changes 4.5% of the original community structure while *NetMelt*$^+$ changes 13.6% of the original community structure.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Data mining; E.1 [**Data Structures**]: Graphs and networks; J.4 [**Computer Applications**]: Social and behavioral sciences

This paper was presented at the First International Workshop on Population Informatics for Big Data (PopInfo'15), Sydney, 10 August 2015.

Keywords

Dissemination control in a population, community structure, graph mining

1. INTRODUCTION

We address the following problem: given a population represented as a graph G,[1] can the dissemination of an entity (such as a meme or a virus) be minimized on G while maintaining G's community structure (where nodes within a community have dense connectivity amongst each other, but they have sparse connectivity with others outside their community)? The problem of controlling an entity's spread on a graph has been studied extensively [8, 10, 3, 18, 21, 14, 20], but (to the best of our knowledge) no one has investigated this problem under the constraint of maintaining the graph's community structure as much as possible. Preserving communities in a graph is an important problem in many real-world applications−e.g., individuals trust members of their communities more than non-members because their interactions are more embedded (due to higher link density between members of a community than to members outside the community) [2].

The *epidemic tipping point* (i.e., whether a dissemination will die out or not) depends on two factors: (a) the entity's strength and (b) the graph's path capacity [3, 18]. We assume that we cannot modify the entity's strength and focus on manipulating the graph's path capacity. However, instead of deleting nodes or links (which affect the graph's community structure), we investigate algorithms that rewire links in order to minimize dissemination *and* minimize change to the community structure of the original (i.e., unperturbed) graph. We quantify minimizing dissemination by the *drop in the largest (in module) eigenvalue of the adjacency matrix*; and measure the amount of change to the community structure of the original (i.e., unperturbed) graph by the *variation of information*, an entropy-based distance function. As we will demonstrate, it is impossible to satisfy both of these minimizations at the same time via the *edge rewire operation*. Thus, we focus on solving a realizable problem−namely, how can we efficiently rewire a set of K edges that effectively contain dissemination and maintain community structure.

To solve the aforementioned problem, we present the *CRlink* algorithm (short for Community Relink), which rewires edges in the graph that lead to the largest drop in the leading eigenvalue of the adjacency matrix by choosing the $relink\text{-}to$ edge with the smallest *eigenscore* within a given community. Furthermore, we present the *constrCRlink* algorithm (short for Constraint Community Relink), which is based on *CRlink* but the rewiring of the edges is based

[1]We use the following terms interchangeably in this paper: graph and network, vertex and node, edge and link.

on node-degree constraints. Experiments on a range of different graphs demonstrate the efficiency and effectiveness of *CRlink* and *constrCRlink*.

The main contributions of the paper are summarized as follows:

- We introduce the problem of minimizing dissemination while preserving community structure on graphs.
- We propose two efficient and effective algorithms for the aforementioned problem—namely, *CRlink* and *constrCRlink*.
- Experimental results on various real graphs show that our *CRlink* and *constrCRlink* algorithms efficaciously solve in the aforementioned problem.

The rest of paper is organized as follows. Section 2 formally defines the edge rewire manipulation and the new problem of minimizing dissemination on a graph while maintaining the graph's community structure. Section 3 proposes algorithms to solve the problem. Section 4 presents our experiments. Section 5 reviews related works. The paper concludes in Section 6.

2. PROBLEM DEFINITION

Table 1 lists the symbols used throughout the paper. We represent an undirected unweighted graph by its adjacency matrix, which is denoted by bold upper-case letter $\mathbf{A}$. Bold lower-case letter $\mathbf{c}$ stands for the community-assignment vector of nodes. The greek letters Φ and Ψ are the sets of deleted and added edges in the rewiring process, respectively. The leading eigenvalue of $\mathbf{A}$ is λ. The bold lower-case letters $\mathbf{u}$ and $\mathbf{v}$ denote the left and right eigenvectors corresponding to λ, respectively.

Symbol	Definition and Description
$\mathbf{A}$	the adjacency matrix of a graph
$\mathbf{A}(i,j)$	the $(i,j)^{\text{th}}$ element of $\mathbf{A}$
$\mathbf{c}$	community-assignment vector of nodes
$\mathbf{c}(i)$	community assignment of node i
Φ	set of deleted edges in rewiring process
Ψ	set of added edges in rewiring process
λ	the leading eigenvalue of $\mathbf{A}$
$\mathbf{u}$, $\mathbf{v}$	the left eigenvector and right eigenvector corresponding to λ
n	the number of nodes in graph
m	the number of edges in graph
K	the edge budget (i.e., the number of edge manipulations)

Table 1: Symbols used in the paper.

We define rewiring of an edge as a two-step operation: (1) delete an existing edge $e{:}\langle src, end\rangle$ and (2) add a formally nonexistent edge, either $\hat{e}{:}\langle src, des\rangle$ or $\hat{e}{:}\langle end, des\rangle$, where des is a node in the graph. All non-existing edges with one of src or end nodes as endpoints are suitable candidates for edge $\hat{e}$. Recall that we are operating on undirected unweighted graphs. Figure 1 depicts the edge rewiring operation. Formally, we define edge rewiring as follows:

DEFINITION 1. ***(Edge Rewiring)**. Given an edge $e{:}\langle src, end\rangle$, an edge rewiring on e deletes e and adds a new edge $\hat{e}$ where $\hat{e}$ is either $\langle src, des\rangle$ or $\langle end, des\rangle$.*

Given the above definition, it is useful to further define two types of edges and three kinds of nodes that participate in the edge rewiring operation. They are as follows: The `rewire-from` edge (denoted by `rf`) is the deleted edge in the rewiring operation, as in edge $e{:}\langle src, end\rangle$ in Figure 1. The `rewire-to` edge (denoted by `rt`) is the newly added edge in the rewiring operation, as in edge $\hat{e}{:}\langle src, des\rangle$ in Figure 1. The *source* node (denoted by src) is the node which is an endpoint in both the `rf` and `rt` edges, as in the node src in Figure 1. The *end* node (denoted by end) is the `rewire-from` node, which appears only in the `rf` edge, as in the node end in Figure 1. The *destination* node (denoted by des) is the `rewire-to` node, which appears only in the `rt` edge, as in the node des in Figure 1.

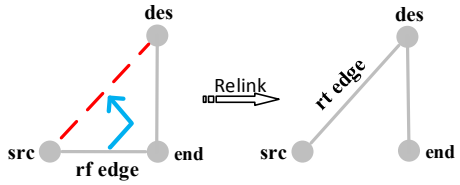

Figure 1: Example of an edge relink. Edge $\langle src, end\rangle$ is deleted and node src is relinked to node des. The edge between src and des is a new edge in the graph.

In order to design an algorithm for minimizing dissemination while preserving community structure, we need to quantify how we measure the decrease in dissemination and the preservation of community structure. For the former, Chakrabarti *et al.* [3] and Prakash *et al.* [18] show that the dissemination process disappears in a graph if the strength of the entity (measured by the ratio of its birth rate α over its death rate β) is less than one over the leading eigenvalue λ of $\mathbf{A}$—i.e., $\alpha/\beta < 1/\lambda$. In other words, λ is the only graph-based parameter that determines the tipping point of the dissemination process. The larger the λ, the smaller the dissemination threshold for the entity to spread out. Thus, an ideal strategy for minimizing dissemination on a graph is to *minimize the leading eigenvalue* λ; or alternatively *maximize the drop in the leading eigenvalue* λ. Tong *et al.* [21] estimate the effects of edge removal on λ via an *eigenscore* function. Specifically, they define the eigenscore of an edge $e{:}\langle i, j\rangle$ as the product of the i-th and j-th elements of the left and right eigenvectors corresponding to λ. We use the eigenscore function to select the `rf`-type edges to be deleted and `rt`-type edges to be added. An `rf`-type edge has the largest eigenscore in the graph. An `rt`-type edge has the smallest eigenscore in the graph. Together these two identify the edge whose rewiring will result in the largest decrease in λ. In addition, we need a way of quantifying how much the community structure of a graph changes as its edges are manipulated. Among the many ways of measuring this quantity, we select the *variation of information* $V(\mathbf{c}, \hat{\mathbf{c}})$ [9]. $V(\mathbf{c}, \hat{\mathbf{c}})$ is a symmetric entropy-based distance function. It measures the "robustness" of a community structure to perturbations in the adjacency matrix. The formal definition of $V(\mathbf{c}, \hat{\mathbf{c}})$ is given in Section 4.1.2. The value of $\frac{V(\mathbf{c},\hat{\mathbf{c}})}{\log n}$ is between 0 (no change) to 1 (complete change), inclusive.

Finding the set of K edges whose deletion maximizes the drop in λ is an NP-hard problem [21]. The most effective approximate edge-deletion algorithm to maximize the drop in λ recomputes the eigenscores of edges after each edge deletion. This approximate algorithm, called *NetMelt*$^+$, is an improved version of *NetMelt* [21]. Edge rewiring, is a combination of edge deletion and edge addition. Under the same budget K, the best case for edge rewiring is to choose K edges of type `rf` to delete as in *NetMelt*$^+$, which leads to maximizing the drop in λ. However, with edge rewiring, there are also K edges of type `rt` that need to be added. These edge additions lead to an increase in λ. Hence, the drop in λ under edge

rewiring is always less than the drop under edge deletion. That is, it is impossible to maximize the drop in λ with edge relinkage, whose edge additions are required to minimize $V(\mathbf{c}, \hat{\mathbf{c}})$.

With above analysis, we look for edges that produce a large drop in the λ and a small value of $V(\mathbf{c}, \hat{\mathbf{c}})$. Thus, the problem is formally defined as follows:

PROBLEM 1. ***Given*** *a graph* $\mathbf{A}$ *and an integer (budget)* K*,* ***output*** *a set of* K_d *edges of type* `rf` *to be deleted from* $\mathbf{A}$ *and a set of* K_a *new edges of type* `rt` *to be added to* $\mathbf{A}$*, which produce a large drop in* λ *and a small value of* $V(\mathbf{c}, \hat{\mathbf{c}})$*. The budget* K *is equal to* K_d *and* $K_a \leqslant K_d$*.*

Note that there may be no associated `rt`-type edge added for a given `rf`-type edge deleted (i.e., $K_a \leqslant K_d$). In the following section, we introduce two algorithms to solve Problem 1.

3. PROPOSED ALGORITHM

In this section, we propose two different strategies and corresponding algorithms to solve Problem 1. In addition, we present an analysis of the complexities of the proposed algorithms.

3.1 Proposed Algorithm: Community Relink (CRlink)

To get the largest drop in λ with edge rewiring (see Definition 1), one can delete K_d edges of type `rf` with the highest eigenscores and add K_a previously non-existent edges of type `rt` with the lowest eigenscores. We name this simple strategy *GRlink* (short for Global Relink). Thus, *GRlink* repeatedly deletes the edge with the highest eigenscore in the graph and adds the edge with the lowest eigenscore from one of the endpoints of the deleted edge to any node in the graph.

The motivation for edge rewiring (i.e., deletion of an existing edge followed addition of a new edge) is to maintain the graph's community structure. The key issue is which previously non-existent edges of type `rt` are suitable for addition. *GRlink* chooses the `rt` edge with the smallest eigenscore in the whole graph. However, from the community structure perspective, edge rewiring among all nodes in the graph may completely change the community structure because it may decrease the connections among nodes within a community while increasing the connections across communities, which can lead to different outcomes for community assignments. Thus, we implement edge rewiring within a community based on the following considerations:

- Both endpoints of most `rf` edges are in the same community (i.e., most `rf` edges are "non-bridges"). Figure 2 reports the ratios of non-bridge edges in various networks.[2] In each network, this ratio is above 80%.
- Edge rewiring in the same community is more effective for maintaining community structure than edge rewiring throughout the whole graph.
- In real applications, it is more realizable for an individual to connect to another individual who is in the same community.

Algorithm 1 describes the Community Relink (*CRlink*) algorithm. In each loop of *CRlink*, it first chooses the `rf` edge with the highest *eigenscore* to delete and then finds the suitable `rt` candidate edges whose `des` node is in the same community with `src` node. Finally, it selects the best `rt` edge with the lowest *eigenscore* among these candidates to add. In some loops of *CRlink*, there may be no associated `rt` edge for an `rf` edge due to the within community constraint. Nonetheless, *CRlink* deletes the `rf` edges in these loops. Thus, $K_a \leqslant K_d$ in the *CRlink* algorithm. Note that newly added edges in former steps can not be re-deleted in later steps, as well as newly deleted edges can not be re-added. Thus in Step 6 of Algorithm 1, $e_{ij} \notin \Psi$ avoids the re-deletion of newly added edges. Step 17 and 21 do not update the deleted (`rf`) edges in $\mathbf{A}$, which guarantees that newly deleted edges will not be re-added.

[2]See Section 4.1.1 for a description of these networks.

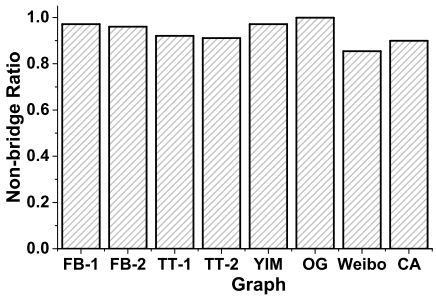

Figure 2: Non-bridge edge ratio of different graphs. Most of the edges in the graphs are non-bridge edges with both endpoints in the same community. (See Section 4.1.1 for a description of the graphs.)

3.2 Proposed Algorithm: Constraint Community Relink (constrCRlink)

CRlink rewires edges by deleting the edges of type `rf` with the largest eigenscores and adding the within community edges of type `rt` with the smallest eigenscores. Here, we investigate the choice of the within community edge `rt` to add.

Some of the most popular community detection algorithms are based on graph modularity [6]. Change in modularity influences the community assignment of each node directly. So we need to figure out the key parameter of a node that correlates with modularity change when edge rewiring happens in the community. Lemma 1 shows that the change in modularity is related *only* to the degree of node des. Recall that des is one of the endpoints of the edge being added that was not present before. The other endpoint is from the deleted edge.

LEMMA 1. *The change in modularity is only related to the node degree of* des *when edge rewiring happens within the same community.*

PROOF. The definition of graph modularity is as follows [6]:

$$Q = \frac{1}{2m} \sum_{i,j} \left[\mathbf{A}[i,j] - \frac{k_i k_j}{2m} \right] \delta(\mathbf{c}(i), \mathbf{c}(j))$$

where k_i is the degree of node i and $\delta(\mathbf{c}(i), \mathbf{c}(j))$ is the community-assignment indicator function of the two nodes i and j. $\delta(\mathbf{c}(i), \mathbf{c}(j))$ is 1 when $\mathbf{c}(i)$ and $\mathbf{c}(j)$ are the same; and 0 otherwise. Let ΔQ_- and ΔQ_+ represent the modularity change of deleting edge $e{:}\langle \hat{i}, \hat{j} \rangle$ and adding edge $e{:}\langle \bar{i}, \bar{j} \rangle$ in the same community, respectively. Hence:

$$\Delta Q_- = \frac{1}{2m} \left[\mathbf{A}[\hat{i}, \hat{j}] - \frac{k_{\hat{i}} k_{\hat{j}}}{2m} \right] \delta(\mathbf{c}(\hat{i}), \mathbf{c}(\hat{j}))$$

$$\Delta Q_+ = \frac{1}{2m} \left[\mathbf{A}[\bar{i}, \bar{j}] - \frac{k_{\bar{i}} k_{\bar{j}}}{2m} \right] \delta(\mathbf{c}(\bar{i}), \mathbf{c}(\bar{j}))$$

Algorithm 1 Community Relink (a.k.a. *CRlink*)

Input: Adjacency matrix $\mathbf{A}$, budget K, community vector $\mathbf{c}$;
Output: K_d deleted edges of type `rf` indexed by set Φ, and a corresponding K_a added edges of type `rt` indexed by set Ψ ($K_a \leqslant K_d = K$);

1: initialize Φ and Ψ to the empty set;
2: **for** $t = 0, 1, ..., K$ **do**
3: compute the leading eigenvalue λ of $\mathbf{A}$;
4: compute the corresponding eigenvectors: $\mathbf{u}$ and $\mathbf{v}$;
5: $score(e_{ij})$=$\mathbf{u}(i)\mathbf{v}(j)$ for $i, j = 1, 2, ..., n$;
6: find $e_{del} = e_{\hat{i}\hat{j}} = argmax_{e_{ij}} score(e_{ij})$, where $e_{ij} \notin \Phi$ and $e_{ij} \notin \Psi$;
7: add the edge e_{del} into Φ;
8: **for** $k = 0, 1, ..., n$ **do**
9: **if** $\mathbf{c}(\hat{i}) == \mathbf{c}(k)$&&$\mathbf{A}[\hat{i}, k] == 0$ **then**
10: $score(\hat{e}_{\hat{i}k}) = \mathbf{u}(\hat{i})\mathbf{v}(k)$;
11: **end if**
12: **if** $\mathbf{c}(\hat{j}) == \mathbf{c}(k)$&&$\mathbf{A}[\hat{j}, k] == 0$ **then**
13: $score(\hat{e}_{\hat{j}k}) = \mathbf{u}(\hat{j})\mathbf{v}(k)$;
14: **end if**
15: **end for**
16: **if** $\hat{e}_{\hat{i}k} \cup \hat{e}_{\hat{j}k} == \emptyset$ **then**
17: $\hat{e}_{add} = null$ and do not update $\mathbf{A}$
18: **else**
19: find $\hat{e}_{add} = argmin_{(\hat{e}_{\hat{i}k} \cup \hat{e}_{\hat{j}k})} score(\hat{e}_{\hat{i}k} \cup \hat{e}_{\hat{j}k})$, where $\hat{e}_{\hat{i}k} \cup \hat{e}_{\hat{j}k} \notin \Psi$;
20: add the new edge $\hat{e}_{add}$ to Ψ;
21: update added (`rt`) edges in $\mathbf{A}$;
22: **end if**
23: **end for**

The overall modularity change $|\Delta Q|$ is:

$$|\Delta Q| = \frac{1}{2m} \frac{k_{\hat{i}}}{2m} \left| k_{\hat{j}} - k_{\bar{j}} \right|$$

where $k_{\hat{i}}$ and $k_{\hat{j}}$ are the degrees of the nodes $\hat{i}$ and $\hat{j}$, which are selected by eigenscore decomposition (see Section 2). Therefore, ΔQ is correlated with the node degree ($k_{\bar{j}}$) of des. □

According to Lemma 1, we should consider the node degree of des when choosing the `rt` edge to add. An intuitive way is to constrain the degree of des node in edge rewiring within community. Adding an edge to a node with small degree impacts the community structure more than adding an edge to a node with large degree. With such consideration, we present the Constraint Community Relink (or *constrCRlink*) algorithm based on *CRlink*. In each iteration of *constrCRlink*, it chooses the `rf` edge with the highest eigenscore to delete; and rewires one of the endpoints to the corresponding lowest eigenscore `rt` edge with a small degree des node. Similar to *CRlink*, $K_a \leqslant K_d$ in *constrCRlink*. We only need to change Step 9 and Step 12 in Algorithm 1 to get the algorithm for *constrCRlink*.

- Step 9: **if** $\mathbf{c}(\hat{i}) == \mathbf{c}(k)$ && $\mathbf{A}[\hat{i}, k] == 0$ && $d_k \leq \rho$ **then** ...
- Step 12: **if** $\mathbf{c}(\hat{j}) == \mathbf{c}(k)$ && $\mathbf{A}[\hat{j}, k] == 0$ && $d_k \leq \rho$ **then** ...

ρ is a small value parameter for degree constraint and d_k denotes the degree of node k. Note that Lemma 1 only considers the case where both endpoints of the deleted edges are in the same community. Thus, *constrCRlink* does not consider the special case where the two endpoints are in different communities. This decision is due to the following two reasons. First, as Figure 2 shows, most of the edges in a given graph are non-bridge edges (i.e., with the two endpoints in the same community). Thus, the case where the two endpoints are in different communities has little influence and so *constrCRlink* ignores it. Second, there are a few deleted edges with two endpoints in different communities. This leads to a more stable community structure since edges across communities are deleted while edges within communities are added.

3.3 Algorithm Complexity Analysis

LEMMA 2. *The time complexities of CRlink and constrCRlink are $\mathcal{O}(K(m+n))$. The space costs of CRlink and constrCRlink are $\mathcal{O}(n^2)$.*

PROOF. In *CRlink* and *constrCRlink*, Steps 3 and 4 take $\mathcal{O}(m+n)$ by Lanczos algorithm [13]. Steps 5 and 6 cost $\mathcal{O}$(m). The loop from Steps 8 to 15 takes $\mathcal{O}(n)$. Step 19 costs $\mathcal{O}(n)$. Over K iterations, the algorithm takes $\mathcal{O}(K(m+n+m+n+n))$ time. Thus, the time complexities of *CRlink* and *constrCRlink* are $\mathcal{O}(K(m+n)$. In many real graphs, $m \sim n \log n$.

In terms of space, we first need $\mathcal{O}(m)$ to store the original graph $\mathbf{A}$. It costs $\mathcal{O}(1)$ and $\mathcal{O}(2n)$ to store the largest eigenvalue and its associated eigenvectors, respectively. In Step 5, it costs $\mathcal{O}(m)$ to store the eigenscores of all edges. Moreover, in the worst case, we need an additional $\mathcal{O}(\binom{n}{2} - m)$ to store the eigenscores of non-existing edges. The storage of deleted edges and added edges take $\mathcal{O}(K)$. Therefore, the total space cost of *CRlink* and *constrCRlink* are $\mathcal{O}(m+1+2n+m+\binom{n}{2}-m+K) \sim \mathcal{O}(m+n+n^2+K)$. Since $n^2 \gg m > n > K$, the total space cost of *CRlink* and *constrCRlink* is $\mathcal{O}(n^2)$ in the worst case. □

4. EXPERIMENTS

This section is divided into three parts: experimental setup, evaluation results, and discussion.

4.1 Experimental Setup

4.1.1 Datasets

Table 2 lists the graphs used in our experiment. All of them are transformed to undirected and unweighted graphs. We use the following six different types of graphs to evaluate our algorithms:[3]

- **Facebook user-postings (FB)**: We use two graphs of this type. Each node represents a Facebook user. An edge between two users means a "posting" event between them.
- **Twitter re-tweet (TT)**: We use two graphs of this type. A node is a Twitter acount. There is an edge between two accounts if a re-tweet event happens between them.
- **Yahoo! Instant Messenger (YIM)**: A node is a Yahoo! IM user. An edge indicates a communication between two users.
- **Oregon Autonomous System (OG)**: A node represents an autonomous system. An edge is a connection inferred from the Oregon route-views.
- **Weibo re-tweet (Weibo)**: A node denotes a Sina-Weibo user. There is an edge between two users if a re-tweet event happens between them.

[3]Most of our datasets are available at `https://snap.stanford.edu/data/`.

- **Collaboration Network of ArXiv (CA)**: Nodes represent scientists, edges represent collaborations (i.e., co-authoring a paper).

Dataset	# of Nodes (n)	# of Edges (m)	# of Communities
FB-1	27,168	26,231	2,154
FB-2	29,557	29,497	1,865
TT-1	25,843	28,124	2,983
TT-2	39,546	45,149	3,920
YIM	50,576	79,219	2,867
OG	7,352	15,665	38
Weibo	34,866	37,849	4,786
CA	5,243	14,484	392

Table 2: Datasets used in our experiments. We use the *Louvain method* [1] to find communities. The number of communities is computed automatically by the Louvain method.

4.1.2 Evaluation Measures

We consider performances on both the decrease in the leading eigenvalue λ and the change in the community structure $V(\mathbf{c}, \hat{\mathbf{c}})$. Given the original graph $\mathbf{A}$ and the perturbed graph $\hat{\mathbf{A}}$, we have the following evaluation measures:

- **Drop in the leading eigenvalue**: We define the percent drop in the leading eigenvalue λ as:

$$\Delta\lambda\% = \frac{100 \times (\lambda - \hat{\lambda})}{\lambda}$$

  where $\hat{\lambda}$ is the leading eigenvalue of $\hat{\mathbf{A}}$. The higher the $\Delta\lambda\%$, the better the performance.

- **Change in the community structure**: We use the *variation of information* $V(\mathbf{c}, \hat{\mathbf{c}})$ [9] between the community structures of $\mathbf{A}$ and $\hat{\mathbf{A}}$ since it has all the properties of a proper distance measure. $V(X, Y)$ is defined as:

$$V(X,Y) = H(X|Y) + H(Y|X)$$
$$= -\sum_{xy} P(x,y) \log \frac{P(x,y)}{P(y)} - \sum_{xy} P(x,y) \log \frac{P(x,y)}{P(x)}$$

  where $H(X|Y)$ and $H(Y|X)$ are conditional entropies. $P(x,y) = n_{xy}/n$, $P(x) = n_x/n$ and $P(y) = n_y/n$, where x and y are the community assignments in $\mathbf{c}$ and $\hat{\mathbf{c}}$, respectively. n_{xy} is the number of nodes which belong to community x in $\mathbf{c}$ and community y in $\hat{\mathbf{c}}$. In addition, we normalized $V(\mathbf{c}, \hat{\mathbf{c}})$ by $1/\log n$ since $\log n$ is the maximum value of $V(\mathbf{c}, \hat{\mathbf{c}})$. The lower the $V(\mathbf{c}, \hat{\mathbf{c}})$, the better the performance (i.e., the more the original community structure is preserved).

To find communities, we use the Louvain method [1] due to its good performance in both efficacy and efficiency. The choice of community discovery algorithm is orthogonal to our work.

4.1.3 Comparison Methods

We compare the results of the following six methods:

1. ***GRlink***: edge rewiring, `rt` edges selected from the whole graph based on eigenscore computation.
2. ***CRlink***: edge rewiring, `rt` edges chosen from within a community based on eigenscore computation.
3. ***constrCRlink***: edge rewiring, `rt` edges selected from within a community based on eigenscore computation and degree constraint $\rho = 1$.
4. ***NetMelt*** [21]: edge deletion, deleted edges selected based on eigenscore computation.
5. ***NetMelt$^+$***: edge deletion, an improved version of $NetMelt$, which re-computes the eigenscore after each edge deletion.
6. ***RandMelt***: edge deletion, deleted edges are chosen randomly.[4]

4.2 Experimental Results

4.2.1 Performance w.r.t. $\Delta\lambda\%$ and $V(\mathbf{c}, \hat{\mathbf{c}})$

First, we evaluate the effectiveness of the different methods, in terms of $\Delta\lambda\%$ and $V(\mathbf{c}, \hat{\mathbf{c}})$, across various edge budgets K. Figure 3 shows that *constrCRlink* performs well in terms of $\Delta\lambda\%$ (it is close to *NetMelt$^+$*, which solely optimizes for $\Delta\lambda\%$) and has the smallest $V(\mathbf{c}, \hat{\mathbf{c}})$. *CRlink* also has good performances in both $\Delta\lambda\%$ and $V(\mathbf{c}, \hat{\mathbf{c}})$. So, our algorithms not only have strong impact in containing dissemination but also maintaining community structure. In addition, as discussed in Section 3, *GRlink* has a large value in $V(\mathbf{c}, \hat{\mathbf{c}})$−i.e., it performs badly in preserving community structure.

Table 3 lists the $\Delta\lambda\%$ and $V(\mathbf{c}, \hat{\mathbf{c}})$ results of different methods with a fixed budget $P = 100 \times \frac{K}{m} \approx 8\%$ across various graphs. The take-away points from this table are the following:

- On average, *constrCRlink* preserves 98.6% of *NetMelt$^+$*'s efficacy in term of $\Delta\lambda\%$.
- On average, *constrCRlink* changes the community structure by 4.5%, while *NetMelt$^+$* changes it by 13.6%. In other words, *NetMelt$^+$* changes the graph's community structure on average about 3 times more than *constrCRlink*.
- As expected, $V(\mathbf{c}, \hat{\mathbf{c}})$ of *GRlink* is the largest among all methods because it is agnostic of a node's community structure when it performs edge rewiring.

There are two seemingly counter-intuitive phenomena in Figure 3 and Table 3. One is that *CRlink* seems to not be as good as *constrCRlink* in $\Delta\lambda\%$, even though *CRlink* has more choices for adding edges. These two methods use different strategies to manipulate the network structure, which result in very different eigenscores. The smallest eigenscore of an edge after *constrCRlink* can be less than the smallest eigenscore of an edge after *CRlink*. The same argument holds when contrasting *GRlink* with *constrCRlink*. The other counter-intuitive phenomenon is that when more edges are modified, $V(\mathbf{c}, \hat{\mathbf{c}})$ of most methods increase as expected, but those of *CRlink* and *constrCRlink* keep decreasing. The reason for this is because edge-deletion methods and *GRlink* tend to change the community structure more as the edge budget increases, which lead to an increase in the community variation of information. However, *CRlink* and *constrCRlink* rewire edges within communities. This (often) makes the community structure more stable as the edge budget increases, which leads to a decrease in the community variation of information.

[4]For *RandMelt*, we run the experiment 100 times and report the average results.

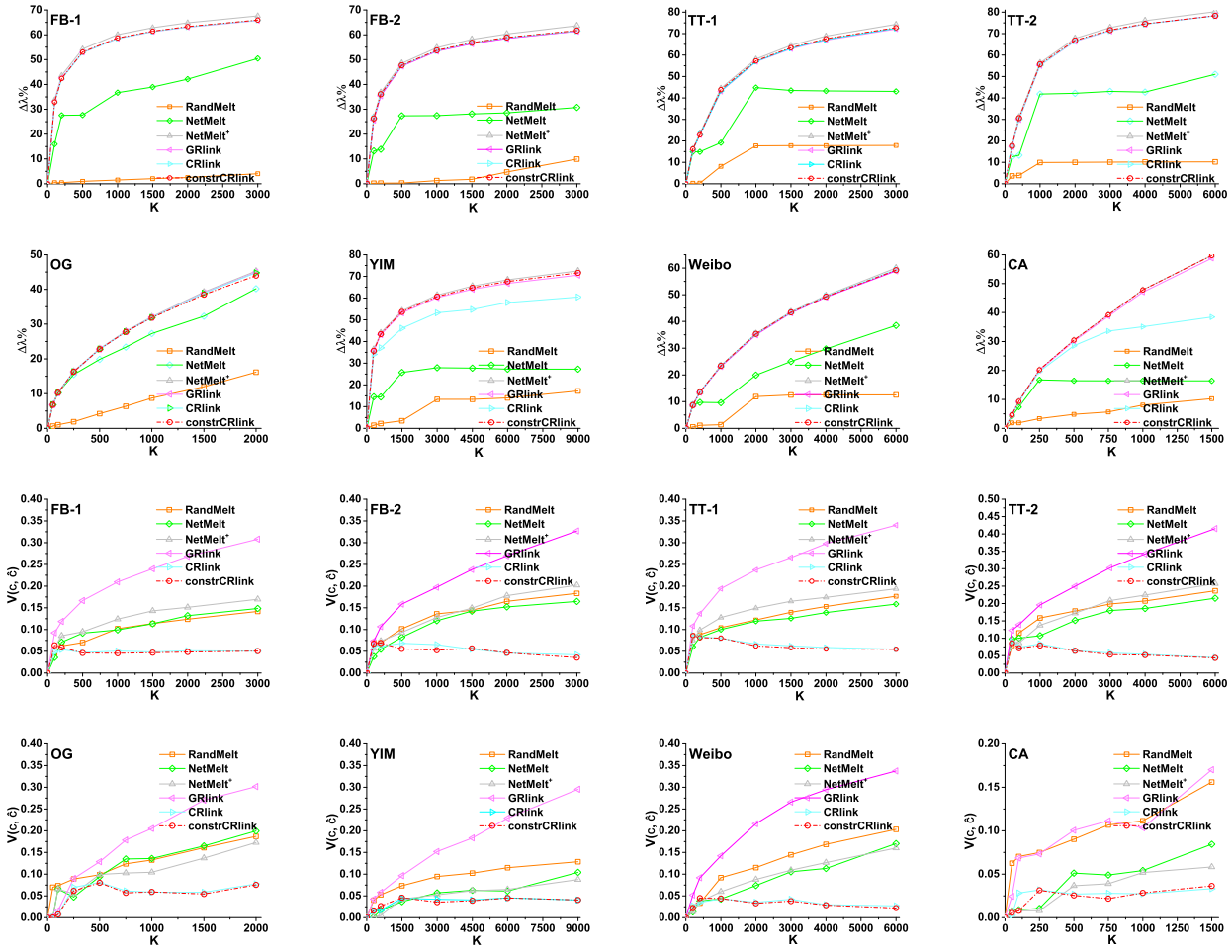

Figure 3: (Best viewed in color.) $\Delta\lambda\%$ and $V(\mathbf{c}, \hat{\mathbf{c}})$ vs. budget K across different graphs. *constrCRlink*'s $\Delta\lambda\%$ closely shadows that of *NetMelt*$^+$ across various graphs (the first and second rows); but its $V(\mathbf{c}, \hat{\mathbf{c}})$ is the smallest (the third and fourth rows). (Note: The x-axes are in different scales due to different graph sizes.)

Dataset	*EvaluationMetric*	*RandMelt*	*NetMelt*	*NetMelt*$^+$	*GRlink*	*CRlink*	*constrCRlink*
FB-1	$\Delta\lambda\%$	2.5228	42.118	**64.842**	63.024	63.132	**63.324**
	$V(\mathbf{c}, \hat{\mathbf{c}})$	0.1239	0.1319	**0.1511**	0.2682	0.0510	**0.0483**
FB-2	$\Delta\lambda\%$	4.7317	28.568	**60.312**	58.521	58.798	**58.902**
	$V(\mathbf{c}, \hat{\mathbf{c}})$	0.1530	0.1390	**0.1741**	0.2974	0.0587	**0.05525**
TT-1	$\Delta\lambda\%$	17.803	43.277	**68.820**	66.946	67.257	**67.592**
	$V(\mathbf{c}, \hat{\mathbf{c}})$	0.1648	0.1519	**0.1780**	0.2694	0.0461	**0.0463**
TT-2	$\Delta\lambda\%$	10.161	42.744	**75.985**	74.396	74.428	**74.549**
	$V(\mathbf{c}, \hat{\mathbf{c}})$	0.2073	0.1856	**0.2242**	0.3425	0.0538	**0.0515**
YIM	$\Delta\lambda\%$	14.022	27.282	**68.413**	66.765	57.914	**67.636**
	$V(\mathbf{c}, \hat{\mathbf{c}})$	0.1148	0.0612	**0.0649**	0.2288	0.0454	**0.0452**
OG	$\Delta\lambda\%$	12.001	32.288	**39.227**	38.979	38.875	**38.476**
	$V(\mathbf{c}, \hat{\mathbf{c}})$	0.1609	0.1649	**0.1370**	0.2701	0.0582	**0.0544**
Weibo	$\Delta\lambda\%$	12.489	25.037	**43.639**	43.193	43.382	**43.417**
	$V(\mathbf{c}, \hat{\mathbf{c}})$	0.1446	0.1056	**0.1091**	0.2659	0.0422	**0.0381**
CA	$\Delta\lambda\%$	8.3240	16.429	**47.832**	47.031	35.081	**47.808**
	$V(\mathbf{c}, \hat{\mathbf{c}})$	0.1114	0.0541	**0.0518**	0.1033	0.0274	**0.0274**

Table 3: Results of $\Delta\lambda\%$ and $V(\mathbf{c}, \hat{\mathbf{c}})$ with a fixed budget $P = 100 \times \frac{K}{m} \approx 8\%$. *constrCRlink* preserves on average 98.6% of *NetMelt*$^+$'s efficacy in $\Delta\lambda\%$; and it performs much better in $V(\mathbf{c}, \hat{\mathbf{c}})$. On average, *constrCRlink* changes the community structure by 4.5%, while *NetMelt*$^+$ change it by 13.6%.

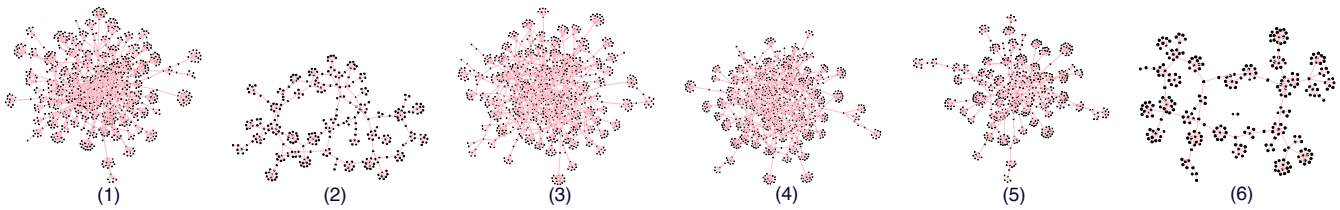

Figure 4: (Best viewed in color.) GCC visualizations of the original/unperturbed graph and the perturbed graphs. (1), (2), (3), (4), (5) and (6) represent the GCCs of the original FB-1 graph, the graph after *GRlink*, the graph after *CRlink*, the graph after *constrCRlink*, the graph after *NetMelt*, and the graph after *NetMelt*$^+$, respectively. GCCs of the graph after *CRlink* (3) and the graph after *constrCRlink* (4) are the most similar to the original FB-1 graph's GCC (1).

4.2.2 *Greatest Community Component Visualization*

To clearly show the differences in the community structure change across different methods, we extract the Greatest Community Component (GCC, which is the community with the maximum number of nodes among all communities) of the original FB-1 graph and the perturbed FB-1 graphs. For better visualization, we use $K = 1300$ (i.e., $P \approx 5\%$). Figure 4 shows that after applying *CRlink* and *constrCRlink*, the GCCs of their (respective) perturbed graphs are similar to the original GCC. After applying *GRlink* and *NetMelt*$^+$, the GCCs of their (respective) perturbed graphs are different from the original GCC (with many nodes having been assigned to other communities). Therefore, from the visualization perspective, *CRlink* and *constrCRlink* perform well in maintaining the community structure.

4.2.3 *Simulation of Virus Propagation*

We evaluate the effectiveness of our algorithms in terms of minimizing the infected population. Specifically, we simulate the SIS (Susceptible-Infected-Susceptible) model [17] of virus propagation. Due to space limitation, we only report the results on the FB-1 graph. The results on the other graphs are similar. In this experiment, we set the budget K to 2000 and the virus strength s to 0.25. Figure 5 reports the relationship between the rate of infected population and the time step. All results are average values of 100 runs. Obviously, the lower the rate, the better the performance in minimizing dissemination. It can be seen that the infected rate of *constrCRlink* is close to *NetMelt*$^+$'s infected rate. This means that *constrCRlink*, as desired, has similar performance to *NetMelt*$^+$ in dissemination minimization. As shown in the previous sections, *constrCRlink* maintains the community structure of the original/unperturbed graph while *NetMelt*$^+$ does not.

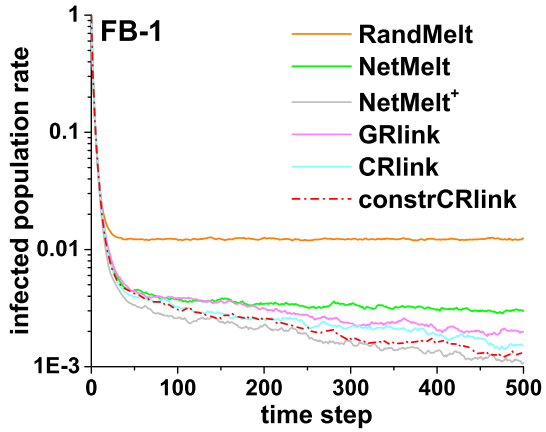

Figure 5: (Best viewed in color.) Comparison of the infected population under the SIS model. Our methods, *CRlink* and *constrCRlink*, have similar infected rates in the population to *NetMelt*$^+$. But, our methods maintain the community structure of the population while *NetMelt*$^+$ does not.

4.3 Analysis and Discussion

We further analyze and discuss the different performances of our methods. For brevity, we only show the analysis on the FB graphs. The other graphs have similar results.

4.3.1 *The impact of parameter ρ in* constrCRlink

We investigate the impact of the node-degree constraint ρ in *constrCRlink*. As discussed in Section 3.2, rewiring different edges impacts the community structure differently. Thus, the parameter ρ plays an important role in *constrCRlink*—namely, it controls which des node the $constrCRlink$ method chooses. Figure 6 illustrates (on average) the relationship between ρ and $V(\mathbf{c}, \hat{\mathbf{c}})$. It can be seen that choosing a des node with smaller degree (i.e., smaller ρ) tends to result in smaller $V(\mathbf{c}, \hat{\mathbf{c}})$; and thus maintains community structure better. Therefore, a smaller ρ should be selected in *constrCRlink*.

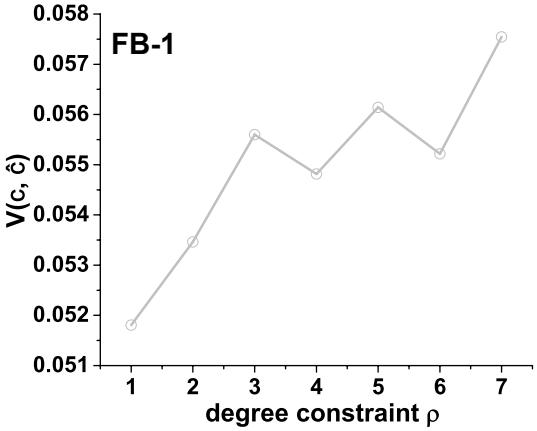

Figure 6: The impact of parameter ρ (i.e., the node-degree constraint) in *constrCRlink*. The smaller the ρ, the smaller the $V(\mathbf{c}, \hat{\mathbf{c}})$ (i.e., less change in the community structure).

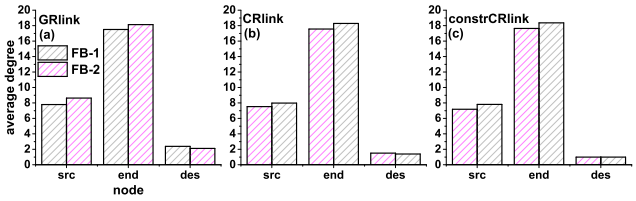

Figure 7: (Best viewed in color.) Average degree of src, end and des nodes selected by our algorithms on the FB-1 and FB-2 graphs. src and end nodes have relatively high degrees, while des nodes have relatively low degrees.

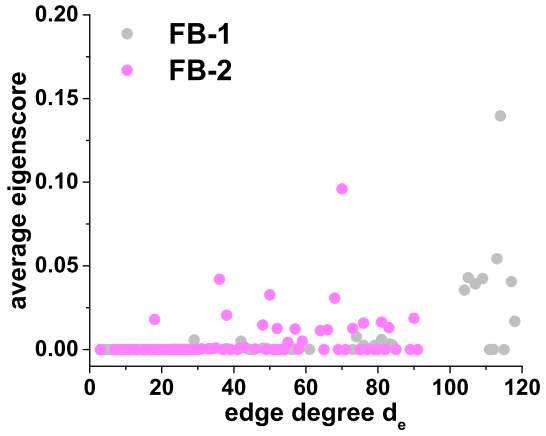

Figure 8: (Best viewed in color.) Relationship between the average eigenscore value and edge degree d_e (which is the sum of the two endpoints' degrees). Larger degree edges tend to have higher eigenscores.

4.3.2 GRlink *vs.* CRlink

According to Figure 3, *GRlink* has the poorest performance in $V(\mathbf{c}, \hat{\mathbf{c}})$. *CRlink* overcomes this weakness via a within community constraint. Before analyzing the performances of them in preserving community structure, we first investigate the degree patterns of src, end, and des nodes. Figures 7(a) and 7(b) report the average degrees of these nodes in *GRlink* and *CRlink* with $K = 2000$. We find that des nodes have low mean degrees and most of them are singletons or connect to one neighbor. src and end nodes have relatively high degrees, which means that we choose `rf` edges with both high degree endpoints and select low degree des nodes in edge rewiring. This is reasonable because most of the highest eigenscore `rf` edges have relatively high degree endpoints and most of the lowest eigenscore `rt` edges should connect relatively low degree nodes, as reported in Figure 8. With these findings, we can assume a common comparison between *GRlink* and *CRlink*, as illustrated in Figure 9.

From Figure 9, *GRlink* chooses `rf` edges with high degree endpoints and selects low degree des nodes outside the community. *CRlink* implements edge rewiring within community and chooses low degree des nodes in the same community. This difference leads to the poor performance of *GRlink* and the good performance of

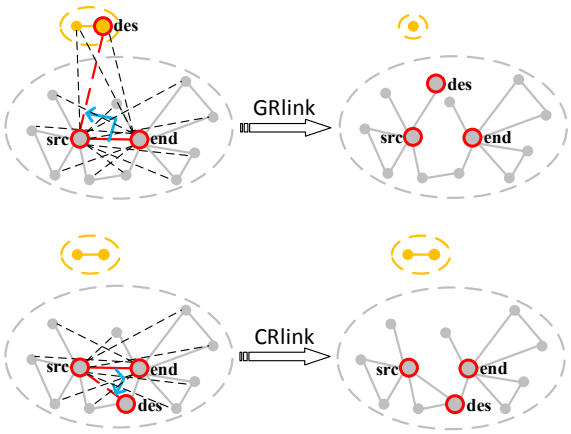

Figure 9: (Best viewed in color.) Comparison case between *GRlink* and *CRlink*. The grey and yellow dashed ellipses represent two communities. The red line and red dashed line denote `rf` and `rt` edges, respectively. The grey dashed lines denote non-existing edge candidates for edge rewiring. The des node is easily reassigned to other communities in *GRlink*. The src and end nodes in *CRlink* have stronger a connection (i.e., a common neighbor des node) than that in *GRlink*.

CRlink in maintaining community structure:

- src and end nodes in *CRlink* hold stronger connections than that in *GRlink*. As shown by Figure 9, the des node becomes a common neighbor of src and end nodes in *CRlink* while the des node is the only neighbor of src node in *GRlink*.
- In *GRlink*, due to low degrees and weak connections with intra-community nodes, the des nodes are easily reassigned to the community of src nodes. See Figure 9, where the des nodes are in a new community in *GRlink*.

To quantify the influences of the aforementioned points in community structure, we define the *Rewiring Community Index* (RCI). For the first point, we define

$$RCI_0 = \sum_{e_{ij} \in \Phi} \frac{I\left[\mathbf{c}(\mathbf{i}) = \mathbf{c}(\mathbf{j})\right] \cdot I\left[\hat{\mathbf{c}}(\mathbf{i}) = \hat{\mathbf{c}}(\mathbf{j})\right]}{I\left[\mathbf{c}(\mathbf{i}) = \mathbf{c}(\mathbf{j})\right]}$$

where $\hat{\mathbf{c}}$ is the community assignment vector of $\hat{A}$. I is the indication function, which equals to 1 when $\mathbf{c}(\mathbf{i}) = \mathbf{c}(\mathbf{j})$ or $\hat{\mathbf{c}}(\mathbf{i}) = \hat{\mathbf{c}}(\mathbf{j})$.

For the second point, we define

$$RCI_1 = \sum_{e_{ij} \in \Psi} \frac{I\left[\mathbf{c}(\mathbf{i}) \neq \mathbf{c}(\mathbf{j})\right] \cdot I\left[\hat{\mathbf{c}}(\mathbf{i}) = \hat{\mathbf{c}}(\mathbf{j})\right]}{I\left[\mathbf{c}(\mathbf{i}) \neq \mathbf{c}(\mathbf{j})\right]}$$

RCI_0 quantifies the similarity of community assignments of `rf` edges' two endpoints (i.e., src and end nodes) between the original graph and the perturbed graph, which is measured in both *GRlink* and *CRlink*. RCI_1 quantifies the community reassignments of des nodes in *GRlink*. Figure 10 (a) and Figure 10 (b) show the results of RCI_0 and RCI_1. From these two plots, RCI_0 of *CRlink* is over 0.4 for most budgets while the corresponding value for *GRlink* is less than 0.1. In addition, RCI_1 of *GRlink* is higher than 0.74. So, we find that *CRlink* has high similarity in community assignments of src and end nodes; consequently it has a good performance in

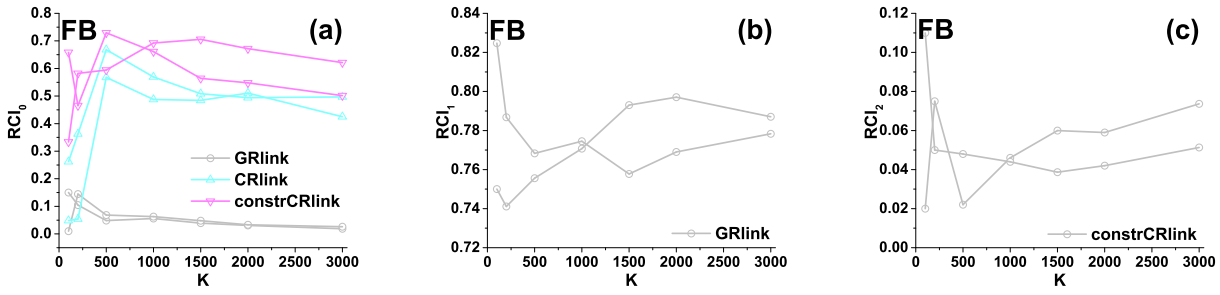

Figure 10: (Best viewed in color.) Rewiring Community Index (RCI) of our methods with different budgets K on the two FB graphs. For simplicity of visualization, the two FB graphs are denoted by the same color lines. (a) *constrCRlink* and *CRlink* have higher RCI_0 than *GRlink*, which means that they are able to keep the same community assignment for the src and end nodes. (b) *GRlink* tends to reassign the community of the des node (i.e., it has high RCI_1). (c) *constrCRlink* is able to maintain the community assignment of the des node (i.e., it has low RCI_2).

maintaining community structure. *GRlink* performs badly in keeping community assignments of src and end nodes, and the community reassignment of des nodes occurs frequently. Both of these lead to the poor performance of *GRlink* on maintaining community structure.

4.3.3 CRlink *vs.* constrCRlink

Results of Section 4.2.1 illustrated that *constrCRlink* performs best in $V(\mathbf{c}, \hat{\mathbf{c}})$, which means that it further improves the performance of *CRlink* by introducing a node-degree constraint (ρ). Similar to the analysis of *GRlink* and *CRlink* discussed in the previous section, we measure the degree patterns of the three kinds of nodes in *constrCRlink*. Figure 7(c) reports that *constrCRlink* also chooses `rf` edges with high degree endpoints and selects within community des nodes with small degree constraint (ρ = 1). In the same way, Figure 11 shows a case comparison between *CRlink* and *constrCRlink*.

In Figure 11, we observe that both *CRlink* and *constrCRlink* choose des nodes in the same community. The difference is that *constrCRlink* only chooses des node whose degree is less than 2. This raises two points, which influence the *constrCRlink*'s performance.

- The src and end nodes keep strong connections in *constrCRlink*. As illustrated in Figure 11, the des nodes become common neighbors of the src and des nodes in *constrCRlink*.
- *constrCRlink* tends to connect within community des nodes, which are weakly connected to the others within the community; and keeps them from being reassigned to other communities. As illustrated in Figure 11, a des node adds one neighbor (src node) within the community in *constrCRlink*.

To quantify influence, we measure the RCI_0 and RCI_2 of *constrCRlink*, where RCI_2 is defined as:

$$RCI_2 = \sum_{e_{ij} \in \Psi} \frac{I\left[\mathbf{c}(\mathbf{i}) = \mathbf{c}(\mathbf{j})\right] \cdot I\left[\hat{\mathbf{c}}(\mathbf{i}) \neq \hat{\mathbf{c}}(\mathbf{j})\right]}{I\left[\mathbf{c}(\mathbf{i}) = \mathbf{c}(\mathbf{j})\right]}$$

RCI_2 quantifies the ability of *constrCRlink* in preventing community reassignments of the des nodes. From the plot in Figure 10(a), we observe that RCI_0 of *constrCRlink* is above 0.5 at most budgets and higher than that of *CRlink*. Very low values (less than 0.12) of RCI_2 in Figure 10(c) demonstrate the strong ability of *constrCRlink* in keeping the des nodes in the same community as before. Both of these result in the good performance of *constrCRlink*.

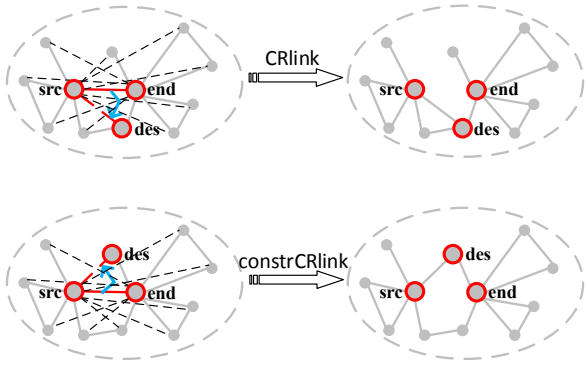

Figure 11: (Best viewed in color.) Case comparison between *CRlink* and *constrCRlink*. The grey dash circle represents a community. The red line and the red dashed line denote `rf` and `rt` edges, respectively. *constrCRlink* keeps the low-degree des node from being reassigned to other communities and holds a strong connection (inn the form of the common neighbor des node) between the src and end nodes.

5. RELATED WORK

The relevant literature for our work can be categorized into two parts: controlling entity dissemination and analyzing community structure.

Controlling entity dissemination. The dynamic processes on large graphs like blogs and propagations [7, 12] are closely related to entity propagation. For the entity dissemination control, Chakrabarti *et al.* [3] and Prakash *et al.* [18] prove that the only graph-based parameter determining the epidemic threshold is the leading eigenvalue of the adjacency matrix of graph. Tong *et al.* [21] introduce the *NetMelt* algorithm, which minimizes the dissemination on a graph by deleting edges with the largest eigenscore associated with the leading eigenvalue (see Section 2 for the definition of

eigenscore). Long *et al.* [14] show that *NetMelt* performs poorly on graphs with small eigen-gaps (like many social graphs) and introduce *MET* (short for Multiple Eigenvalues Tracking) to overcome the small eigen-gap problem. *Chan et al.* [4] track multiple eigenvalues for the purpose of measuring graph robustness. Kuhlman *et al.* [11] study contagion blocking in graphs via edge deletion. Saha *et al.* [20] developed *GreedyWalk* approximation algorithms for reducing the spectral radius by removing the minimum cost set of edges or nodes. To the best of our knowledge, no previous work has investigated edge relinkage in order to minimize dissemination while maintaining community structure.

Analyzing community structure. Besides entity dissemination control, we try to minimize the change in the graph's community structure after perturbation. Many efforts have been devoted to community structure detection and analysis. The past literatures [6, 1, 19, 5] propose several effective methods to detect communities in real-world graphs. Leskovec *et al.* [15] investigate a range of community detection methods in order to understand the difference in their performances. Nematzadeh *et al.* [16] investigate the impact of community structure on information diffusion with the linear threshold model. Karrer *et al.* [9] study the significance of community structure by measuring its robustness to small perturbations in graph structure. Motivated by this last work, we use the difference in community assignment of each node to quantify the abilities of the different algorithms in preserving the graph's community structure.

6. CONCLUSIONS

We present the problem of minimizing dissemination in a population (that is represented as a complex network) while maintaining its community structure (where community is defined as a group of individuals with more links between them than to outside members). Due to the poor performance of edge deletions in preserving community structure, we introduce the edge-rewiring framework and two algorithms: *CRlink* and *constrCRlink*. *CRlink* tends to rewire edges within a community; *constrCRlink* improves *CRlink*'s performance by adding node-degree constraint to rewired edges. Our experimental results on several real-world graphs show that *CRlink* and *constrCRlink* preserve most of the efficacy (more than 98.6%) of *NetMelt*$^+$ in dissemination minimization. Besides, *CRlink* and *constrCRlink* perform much better in preserving community structure (only 4.5% change) than other methods like *NetMelt*$^+$ (with 13.6% change). Furthermore, we investigate the reasons for the different performances of our algorithms.

7. ACKNOWLEDGMENTS

This work was funded in part by LLNL under Contract DE-AC52-07NA27344, by NSF CNS-1314603, by DTRA HDTRA1-10-1-0120, and by DAPRA under SMISC Program Agreement No. W911NF-12-C-0028.

8. REFERENCES

[1] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

[2] R. S. Burt. *Structural Holes: The Social Structure of Competition*. Harvard University Press, 1992.

[3] D. Chakrabarti, Y. Wang, C. Wang, J. Leskovec, and C. Faloutsos. Epidemic thresholds in real networks. *ACM Trans. on Info. and Sys. Sec.*, 10(4):1, 2008.

[4] H. Chan, L. Akoglu, and H. Tong. Make it or break it: Manipulating robustness in large networks. In *SDM*, pages 325–333, 2014.

[5] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010.

[6] M. Girvan and M. E. Newman. Community structure in social and biological networks. *PNAS USA*, 99(12):7821–7826, 2002.

[7] D. Gruhl, R. Guha, D. Liben-Nowell, and A. Tomkins. Information diffusion through blogspace. In *WWW*, pages 491–501, 2004.

[8] H. W. Hethcote. The mathematics of infectious diseases. *SIAM review*, 42(4):599–653, 2000.

[9] B. Karrer, E. Levina, and M. E. Newman. Robustness of community structure in networks. *Physical Review E*, 77(4):046119, 2008.

[10] D. Kempe, J. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *KDD*, pages 137–146, 2003.

[11] C. J. Kuhlman, G. Tuli, S. Swarup, M. V. Marathe, and S. Ravi. Blocking simple and complex contagion by edge removal. In *ICDM*, pages 399–408, 2013.

[12] R. Kumar, J. Novak, P. Raghavan, and A. Tomkins. On the bursty evolution of blogspace. In *WWW*, pages 159–178, 2005.

[13] C. Lanczos. *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*. United States Gov. Press Office, 1950.

[14] L. T. Le, T. Eliassi-Rad, and H. Tong. MET: A fast algorithm for minimizing propagation in large graphs with small eigen-gaps. In *SDM*, 2015.

[15] J. Leskovec, K. J. Lang, and M. Mahoney. Empirical comparison of algorithms for network community detection. In *WWW*, pages 631–640, 2010.

[16] A. Nematzadeh, E. Ferrara, A. Flammini, and Y.-Y. Ahn. Optimal network modularity for information diffusion. *Physical Review Letters*, 113(8):088701, 2014.

[17] R. Pastor-Satorras and A. Vespignani. Epidemic spreading in scale-free networks. *Physical Review Letters*, 86(14):3200, 2001.

[18] B. A. Prakash, D. Chakrabarti, M. Faloutsos, N. Valler, and C. Faloutsos. Threshold conditions for arbitrary cascade models on arbitrary networks. In *ICDM*, pages 537–546, 2011.

[19] M. Rosvall and C. T. Bergstrom. Maps of random walks on complex networks reveal community structure. *PNAS USA*, 105(4):1118–1123, 2008.

[20] S. Saha, A. Adiga, B. A. Prakash, and A. K. S. Vullikanti. Approximation algorithms for reducing the spectral radius to control epidemic spread. In *SDM*, 2015.

[21] H. Tong, B. A. Prakash, T. Eliassi-Rad, M. Faloutsos, and C. Faloutsos. Gelling, and melting, large graphs by edge manipulation. In *CIKM*, pages 245–254, 2012.