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), 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 edges. Second, we propose two effective and efficient algorithms: CRlink (short for Community Edge ReLink) and constrCRlink (short for Constraint Community Edge 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 edge 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

Keywords

Dissemination control in a population, community structure, graph mining

1. INTRODUCTION

We address the following problem: given a population represented as a graph $G = (V,E)$, 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 edge 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 edges (which affect the graph’s community structure), we investigate algorithms that rewire edges in order to minimize dissemination and minimize change to the community structure of the original/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/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 Edge ReLink), which rewrites edges in the graph that lead to the largest drop in the leading eigenvalue of the adjacency matrix by choosing relink-to edge with the smallest eigenscore within a given community. Furthermore, we present the constrCRlink algorithm (short for Constraint Community Edge ReLink) which is based on CRlink but the rewiring of the edges is based 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 representing populations.
- 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 perform well in the aforementioned problem.

The rest of the paper is organized as follows. Section 2 formally defines edge rewire manipulation and the new problem. Section 3 proposes algorithms to solve it. Section 4 presents our experiments. Section 5 reviews the 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 \( e \) stands for the community-assignment vector of nodes. We use the Greek letters \( \Phi \) and \( \Psi \) to be the sets of deleted edges and added edges in the rewiring process. In addition, we represent the leading eigenvalue of \( \mathbf{A} \) by \( \lambda \). The bold lower-case letters \( u \) and \( v \) denote the left and right eigenvectors corresponding to \( \lambda \), respectively.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathbf{A} )</td>
<td>the adjacency matrix of a graph</td>
</tr>
<tr>
<td>( \mathbf{A}(i,j) )</td>
<td>the ((i,j))\textsuperscript{th} element of ( \mathbf{A} )</td>
</tr>
<tr>
<td>( e )</td>
<td>community-assignment vector of nodes</td>
</tr>
<tr>
<td>( \mathbf{e}(i) )</td>
<td>community assignment of node ( i )</td>
</tr>
<tr>
<td>( \Phi )</td>
<td>set of deleted edges in rewiring process</td>
</tr>
<tr>
<td>( \Psi )</td>
<td>set of added edges in rewiring process</td>
</tr>
<tr>
<td>( \lambda )</td>
<td>the leading eigenvalue of ( \mathbf{A} )</td>
</tr>
<tr>
<td>( u, v )</td>
<td>the left eigenvector and right eigenvector corresponding to ( \lambda )</td>
</tr>
<tr>
<td>( n )</td>
<td>the number of nodes in graph</td>
</tr>
<tr>
<td>( m )</td>
<td>the number of edges in graph</td>
</tr>
<tr>
<td>( K )</td>
<td>the edge budget (the number of edge manipulation)</td>
</tr>
</tbody>
</table>

Table 1: Symbols used in the paper.

We define rewiring of an edge as a two-step operation: (1) delete an existing edge \( e: (\text{src}, \text{end}) \) and (2) add a formally nonexistent edge, either \( \hat{e}: (\text{src}, \text{des}) \) or \( \hat{e}: (\text{end}, \text{des}) \). 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 \( e: (\text{src}, \text{end}) \), an edge rewiring on \( e \) deletes \( e \) and adds a non-existing edge \( \hat{e} \) where \( \hat{e} \) is either \( (\text{src}, \text{des}) \) or \( (\text{end}, \text{des}) \).

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:

- rewired-from edge (denoted by rf): the deleted edge in the rewiring operation, as in edge \( e: (\text{src}, \text{end}) \) in Figure 1.
- rewired-to edge (denoted by rt): the added edge in the rewiring operation, as in edge \( \hat{e}: (\text{src}, \text{des}) \) in Figure 1.
- source node (denoted by src): the node which is an endpoint in both the rf and rt edges, as in the node src in Figure 1.
- end node (denoted by end): the rewired-from node, which appears only in the rf edge, as in the node end in Figure 1.
- destination node (denoted by des): the rewired-to node, which appears only in the rt edge, as in the node des in Figure 1.

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 \( \alpha \) over its death rate \( \beta \)) is less than one over the leading eigenvalue \( \lambda \) of \( \mathbf{A} \)—i.e., \( \alpha/\beta < 1/\lambda \). In other words, \( \lambda \) is the only graph-based parameter that determines the tipping point of the dissemination process. The larger the \( \lambda \), the smaller the dissemination threshold for the entity to spread out. Thus, an ideal strategy for minimizing dissemination on a graph is to maximize the drop in the leading eigenvalue \( \lambda \) or alternatively maximize the drop in the leading eigenvalue \( \lambda \). Tong et al. [21] estimate the effects of edge removal on \( \lambda \) via an eigenscore function. Specifically, they define the eigenscore of an edge \( e: (i,j) \) as the product of the \( i \)-th and \( j \)-th elements of the left and right eigenvectors corresponding to \( \lambda \). 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 \( \lambda \). Now 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 \( V(e, \hat{e}) \) [9]. \( V(e, \hat{e}) \) 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 \( V(e, \hat{e}) \) is given in Section 4.1.2. The value of \( V(e, \hat{e}) \) is between 0 (no change) to 1 (complete change), inclusive.

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:

- rewired-from edge (denoted by rf): the deleted edge in the rewiring operation, as in edge \( e: (\text{src}, \text{end}) \) in Figure 1.

Figure 1: Example of an edge relink. Edge \( (\text{src}, \text{end}) \) is deleted and node src is relinked to the formerly non-existent node des.
to choose \( K \) edges of type \( rt \) to delete as in \( NetMelt^+ \), which leads to maximizing the drop in \( \lambda \). 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 \( \lambda \). Hence, the drop in \( \lambda \) under edge rewiring is always less than the drop under edge deletion. That is, it is impossible to maximize the drop in \( \lambda \) with edge relinkage, whose edge additions are required to minimize \( V(c, \hat{e}) \).

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

**Problem 1.** Given a graph \( A \) and an integer (budget) \( K \), output a set of \( K_d \) edges of type \( rt \) to be deleted from \( A \) and a set of \( K_a \) non-existing edges of type \( rt \) to be added to \( A \), which produce a large drop in \( \lambda \) and a small value of \( V(c, \hat{e}) \). The budget \( K \) is equal to \( K_d \) and \( K_a \leq K_d \).

Note that there may be no associated \( rt \)-type edge added for a given \( rt \)-type edge deleted (i.e., \( K_a \leq K_d \)). In the following section, we introduce two kinds of algorithms to solve Problem 1. Details in the next section of this paper.

### 3. Proposed Algorithm

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

#### 3.1 Proposed Algorithm: Community Edge Relink (CRlink)

To get the largest drop in \( \lambda \), we should choose \( K_d \) edges of type \( rt \) with the highest eigenscores to delete and add \( K_a \) previously non-existent edges of type \( rt \) with the lowest eigenscores. We name this simple strategy GlobalEdgeRelink (or GRelink) algorithm. Thus, in GRelink we delete the edge with the highest eigenscore in the graph and add the edge with the lowest eigenscore from one of the endpoints of the deleted edge to any node in the graph.

To maintain community structure, edge rewiring adds edges. The key point is which previously non-existent edges of type \( rt \) are suitable for addition. GRelink chooses the \( rt \) edge with the smallest eigenscore in the whole graph. However, from the community structure perspective, edge rewiring among all nodes may completely change the community structure because it may decrease the connections among nodes within a community while increasing edges between communities, which can lead to different outcomes for community assignments. Thus, we implement edge rewiring within a community based on the following considerations:

- Figure 2 reports the ratios of non-bridge edges in different networks that we use in this paper (see Section 4.1.1). Most of the \( rf \) edges deleted are non-bridge edges—i.e., both end points of most \( rf \) edges are in the same community.
- 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 a user to friend or follow a user who is in the same community.

Algorithm 1 describes the Community Edge 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 \( rf \) candidate edges whose \( des \) node is in the same community with \( src \) node. Finally, it selects the best \( rf \) edge with the lowest eigenscore among these candidates to add. In some loops of

![Figure 2: Non-bridge edge ratio of different graphs. Most of the edges in the graphs are non-bridge edges. (See Section 4.1.1 for a description of the graphs.)](image)

### 3.2 Proposed Algorithm: Constraint Community Edge 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.** The change in modularity is only related with the node degree of \( des \) when edge rewiring happens within the same community.

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

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

where \( \delta(c(i), c(j)) \) is the community-assignment indicator function of two nodes and equals to 1 when \( c(i) = c(j) \) are the same; and \( k_i, k_j \) is the degree of node \( i \). Let \( \Delta Q_- \) and \( \Delta Q_+ \) represent the modularity change of deleting edge \( c(\hat{i}, \hat{j}) \) and adding edge \( c(\hat{i}, \hat{j}) \) in the same community, respectively. Hence:

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

\[
\Delta Q_+ = \frac{1}{2m} \left[ A[\hat{i}, \hat{j}] - \frac{k_{\hat{i}} k_{\hat{j}}}{m} \right] \delta(c(\hat{i}), c(\hat{j}))
\]
Algorithm 1 Community-Edge-Relink (a.k.a. CRLink)

Input: adjacency matrix $A$, budget $K$, community vector $c$;

Output: $K_a$ deleted edges of type $rf$ indexed by set $\Phi$, and a corresponding $K_a$ added edges of type $rt$ indexed by set $\Psi$ ($K_a \leq K_d = K$);

1: initialize $\Phi$ and $\Psi$ to the empty set;
2: for $t = 0, 1, \ldots, K$ do
3: compute the leading eigenvalue $\lambda$ of $A$;
4: compute the corresponding eigenvectors: $u$ and $v$;
5: $score(e_{ij}) = u(i)v(j)$ for $i, j = 1, 2, \ldots, n$;
6: find $e_{del} = e_{ij} = \arg\max_{e_{ij}} score(e_{ij})$, where $e_{ij} \notin \Phi$ and $e_{ij} \notin \Psi$;
7: add the edge $e_{del}$ into $\Phi$;
8: for $k = 0, 1, \ldots, n$ do
9: if $c(i) = c(k) \&\& A[i, k] == 0$ then
10: $score(\hat{e}_{ik}) = u(i)v(k)$;
11: end if
12: if $c(j) = c(k) \&\& A[j, k] == 0$ then
13: $score(\hat{e}_{jk}) = u(j)v(k)$;
14: end if
15: end for
16: if $\hat{e}_{ik} \cup \hat{e}_{jk} == \emptyset$ then
17: $\hat{e}_{add} = \arg\min(\hat{e}_{ik} \cup \hat{e}_{jk}) \cdot score(\hat{e}_{ik} \cup \hat{e}_{jk})$, where $\hat{e}_{ik} \cup \hat{e}_{jk} \notin \Psi$;
18: add the non-existing edge $\hat{e}_{add}$ into $\Psi$;
19: update added ($rt$) edges in $A$;
20: end if
21: end for
22: end if
23: end for

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

$$|\Delta Q| = \frac{1}{2m} \sum \frac{k_i}{2m} |k_j - k_i|$$

$k_i$ and $k_j$ are decided by eigenscore decomposition. Therefore, $|\Delta Q|$ is correlated with the node degree ($k_j$) of des. \hfill \Box

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 Constraint Community Edge Relink (or constrCRLink) algorithm based on CRLink.

In each loop of constrCRLink, it chooses the $rf$ edge with the highest eigenscore to delete; and rewire it to the corresponding lowest eigenscore $rt$ edge with small degree des node. Similar to CRLink, $K_a \leq K_d$ in constrCRLink. We only need to change Step 9 and Step 12 in Algorithm 1 to get the algorithm for constrCRLink.

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

$\rho$ 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 end nodes of the removed edges are in the same community. Thus, constrCRLink does not consider the special case where two end nodes are in the different communities due to the two main reasons:

- As Figure 2 shows, most of the edges in graphs are non-bridge edges. The case where two end nodes are in different communities has little influence and so we ignore it in constrCRLink.
- There are few removed edges with two end nodes in different communities. This leads to a more stable community structure since edges without communities are deleted while edges within communities are added.

3.3 Algorithm Complexity Analysis

**Lemma 2.** The time complexity of CRLink and constrCRLink are $O(n\log nK)$. The space cost of CRLink and constrCRLink are $O(n\log n + K)$.

**Proof.** In CRLink and constrCRLink algorithms, Steps 3 and 4 take $O(m + n)$ by Lanczos algorithm [13]. Step 5-6 costs $O(m)$. The loop from Steps 8 to 15 takes $O(n')(\max(n') = n)$. Step 19 costs $O(n')$. The overall loop of the algorithm takes $O(K(m + n + n'))$. In addition, $n \sim n \log n$ in real graphs. Thus, the time complexity of CRLink and constrCRLink are $O(n \log nK)$.

In terms of space, we first need $O(n)(O(n \log n))$ to store the original graph $A$ and it costs $O(n)$ and $O(1)$ to store the eigenvectors and eigenvalue, respectively. Then in Step 5, it costs $O(n)$ to store the eigenscores of all edges. Moreover, we need additional $O(n')(\max(n') = n)$ to store eigenscores of specific non-existing edges. Finally, storage of deleted edges and added edges take $O(K)$. Therefore, the total space cost of CRLink and constrCRLink are $O(n \log n + K)$.

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 six different types of graph to evaluate our algorithms including:

- **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 account. 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.

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).

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

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 \( \lambda \) and the change in the community structure \( V(e, \hat{e}) \). Given the original graph \( A \) and the perturbed graph \( \hat{A} \), we have the following evaluation measures:

- **Drop in the leading eigenvalue**: We define the percent drop in the leading eigenvalue \( \lambda \) as:

  \[
  \Delta \lambda \% = 100 \times \frac{\lambda - \hat{\lambda}}{\lambda}
  \]

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

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

  \[
  V(X, Y) = H(X|Y) + H(Y|X)
  \]

  \[
  = - \sum_{x,y} P(x, y) \log \frac{P(x, y)}{P(y)} - \sum_{x,y} P(x, y) \log \frac{P(x, y)}{P(x)}
  \]

  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 \( e \) and \( \hat{e} \), respectively. \( n_{xy} \) is the number of nodes which belong to community \( x \) in \( e \) and community \( y \) in \( \hat{e} \). In addition, we normalized \( V(e, \hat{e}) \) by \( 1/\log n \) since \( \log n \) is the maximum value of \( V(e, \hat{e}) \). The lower the \( V(e, \hat{e}) \), 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 results of six methods including:

- **CRLink**: edge rewiring, \( \tau \) \( \in \) edges chosen within a community based on eigenscore.
- **constrCRLink**: edge rewiring, \( \tau \) \( \in \) edges selected within a community based on eigenscore and degree constraint \( \rho = 1 \).
- **NetMelt** [21]: edge deletion, deleted edges selected based on eigenscore.
- **NetMelt+**: edge deletion, an improved version of NetMelt, which re-computes the eigenscore after each edge deletion.
- **RandMelt**: edge deletion, deleted edges are chosen randomly.

4.2 Experimental Results

4.2.1 The impact of \( \Delta \lambda \% \) and \( V(e, \hat{e}) \)

First, we evaluate the effectiveness of the different methods with different budgets. Figure 3 shows that constrCRLink performs well in \( \Delta \lambda \% \) (close to NetMelt+), which solely optimizes for \( \Delta \lambda \% \) and has the smallest \( V(e, \hat{e}) \). CRLink also has good performances in both \( \Delta \lambda \% \) and \( V(e, \hat{e}) \). These results meet what we expected. Our algorithms not only have strong impact in containing dissemination but also maintaining community structure. In addition, as we discussed in Section 3, GRLink has a large value in \( V(e, \hat{e}) \)—i.e., it performs badly in preserving community structure.

Table 3 shows \( \Delta \lambda \% \) and \( V(e, \hat{e}) \) results with different algorithms on a fixed budget \( P = 100 \times \frac{k}{m} \approx 8\% \) in different graphs. This table informs us of the effectiveness of our algorithms in detail:

- **constrCRLink** has similar results to NetMelt+ in \( \Delta \lambda \% \). On average, constrCRLink preserves 98.6\% of NetMelt+’s efficacy.
- The relative improvements of constrCRLink on \( V(e, \hat{e}) \) are desirable. On average, constrCRLink changes the community structure by 4.5\%, while NetMelt+ change it by 13.6\. NetMelt+ changes community structure percentage about 3 times over constrCRLink.
- \( V(e, \hat{e}) \) of GRLink is the largest among all methods.

There are two seemingly counter-intuitive phenomena in Figure 3 and Table 3. One is that GRLink seems not as good as constrCRLink in \( \Delta \lambda \% \) although CRLink has more choices for adding edges. This is because these two strategies choose different ways to change the network structure, which leads to very different eigenscore results. 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, \( V(e, \hat{e}) \) of most algorithms 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 community structure more with the increment of edge budget, which leads to the increment of community variation of information. CRLink and constrCRLink, however, rewire edges within communities. This (often) makes community structure more stable with the increment in budget, which leads to the decrement of community variation of information.

\(^3\)Here we run the experiment several times and report the average result.
Figure 3: $\Delta \lambda$% and $V(c, \hat{c})$ versus budget $K$ in different graphs. constrCRlink has $\Delta \lambda$% close to NetMelt$^+$ and the smallest $V(c, \hat{c})$ across different budgets. The x-axes are in different scales due to different graphs sizes.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Evaluation Metric</th>
<th>RandMelt</th>
<th>NetMelt</th>
<th>NetMelt$^+$</th>
<th>GRlink</th>
<th>CLink</th>
<th>constrCRlink</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB-1</td>
<td>$\Delta \lambda$%</td>
<td>2.5228</td>
<td>42.118</td>
<td>64.842</td>
<td>63.024</td>
<td>63.132</td>
<td>63.324</td>
</tr>
<tr>
<td></td>
<td>$V(c, \hat{c})$</td>
<td>0.1239</td>
<td>0.1319</td>
<td>0.1511</td>
<td>0.2682</td>
<td>0.0510</td>
<td>0.0483</td>
</tr>
<tr>
<td>FB-2</td>
<td>$\Delta \lambda$%</td>
<td>4.7317</td>
<td>28.568</td>
<td>60.312</td>
<td>58.521</td>
<td>58.798</td>
<td>58.902</td>
</tr>
<tr>
<td></td>
<td>$V(c, \hat{c})$</td>
<td>0.1530</td>
<td>0.1390</td>
<td>0.1741</td>
<td>0.2974</td>
<td>0.0587</td>
<td>0.0525</td>
</tr>
<tr>
<td>TT-1</td>
<td>$\Delta \lambda$%</td>
<td>17.803</td>
<td>43.277</td>
<td>68.820</td>
<td>66.946</td>
<td>67.257</td>
<td>67.592</td>
</tr>
<tr>
<td></td>
<td>$V(c, \hat{c})$</td>
<td>0.1648</td>
<td>0.1519</td>
<td>0.1780</td>
<td>0.2694</td>
<td>0.0461</td>
<td>0.0463</td>
</tr>
<tr>
<td>TT-2</td>
<td>$\Delta \lambda$%</td>
<td>10.161</td>
<td>42.744</td>
<td>75.985</td>
<td>74.396</td>
<td>74.428</td>
<td>74.549</td>
</tr>
<tr>
<td></td>
<td>$V(c, \hat{c})$</td>
<td>0.2073</td>
<td>0.1856</td>
<td>0.2242</td>
<td>0.3425</td>
<td>0.0538</td>
<td>0.0515</td>
</tr>
<tr>
<td>YIM</td>
<td>$\Delta \lambda$%</td>
<td>14.022</td>
<td>27.282</td>
<td>68.413</td>
<td>66.765</td>
<td>57.914</td>
<td>67.636</td>
</tr>
<tr>
<td></td>
<td>$V(c, \hat{c})$</td>
<td>0.1148</td>
<td>0.0612</td>
<td>0.0649</td>
<td>0.2288</td>
<td>0.0454</td>
<td>0.0452</td>
</tr>
<tr>
<td>OG</td>
<td>$\Delta \lambda$%</td>
<td>12.001</td>
<td>32.288</td>
<td>39.227</td>
<td>38.979</td>
<td>38.875</td>
<td>38.476</td>
</tr>
<tr>
<td></td>
<td>$V(c, \hat{c})$</td>
<td>0.1609</td>
<td>0.1649</td>
<td>0.1370</td>
<td>0.2701</td>
<td>0.0582</td>
<td>0.0544</td>
</tr>
<tr>
<td>Weibo</td>
<td>$\Delta \lambda$%</td>
<td>12.489</td>
<td>25.037</td>
<td>43.639</td>
<td>43.193</td>
<td>43.382</td>
<td>43.417</td>
</tr>
<tr>
<td></td>
<td>$V(c, \hat{c})$</td>
<td>0.1446</td>
<td>0.1056</td>
<td>0.1091</td>
<td>0.2659</td>
<td>0.0422</td>
<td>0.0381</td>
</tr>
<tr>
<td>CA</td>
<td>$\Delta \lambda$%</td>
<td>8.324</td>
<td>16.429</td>
<td>47.832</td>
<td>47.031</td>
<td>35.081</td>
<td>47.808</td>
</tr>
<tr>
<td></td>
<td>$V(c, \hat{c})$</td>
<td>0.1114</td>
<td>0.0541</td>
<td>0.0518</td>
<td>0.1033</td>
<td>0.0274</td>
<td>0.0274</td>
</tr>
</tbody>
</table>

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

In order to clearly show the differences in community structure change under different methods, we extract the Greatest Community Component (a.k.a. GCC, which has the maximum number of nodes in 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 $\text{constrCRlink}$ and $\text{CRlink}$, the GCC of the perturbed graphs are similar to the original GCC. After applying $\text{GRlink}$ and $\text{NetMelt}^+$, the GCC of the perturbed graphs are different from the original GCC (with many nodes having been assigned to other communities). Therefore, from the visualization perspective, $\text{CRlink}$ and $\text{constrCRlink}$ algorithms 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 $K = 2000$ and virus strength $\rho = \alpha/\beta \approx 0.25$. Figure 5 reports the relationship between the rate of infected population and the time step. All results are averaged over several times. Obviously, the lower the rate, the better the performance in minimizing dissemination. It can be seen that the infected rate of $\text{constrCRlink}$ is close to $\text{NetMelt}^+$’s infected rate. This means that $\text{constrCRlink}$, as desired, has similar performance to $\text{NetMelt}^+$ in dissemination minimization. As shown previously, our methods maintain community structure of the original/ unperturbed graph while $\text{NetMelt}^+$ does not.

4.3 Analysis and Discussion

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

4.3.1 The impact of parameter $\rho$ in $\text{constrCRlink}$

We investigate the impact of node-degree constraint in $\text{constrCRlink}$. As we discussed in Section 3.2, different edges impact the community structure differently. Thus, the parameter $\rho$ plays an important role in our method. It controls which des node the $\text{constrCRlink}$ algorithm should choose. Figure 6 illustrates (on average) the relationship between $\rho$ and $V(c, \hat{c})$. It can be seen that choosing a des node with smaller degree (i.e., smaller $\rho$) tends to have smaller $V(c, \hat{c})$ and maintains community structure better. Therefore, a smaller $\rho$ should be selected in $\text{constrCRlink}$.

4.3.2 $\text{GRlink}$ vs. $\text{CRlink}$

Figure 4: GCC visualization of the original/unperturbed graph and the perturbed graphs. (1), (2), (3), (4), (5) and (6) represent GCCs of the original FB-1 graph, after $\text{GRlink}$ graph, after $\text{CRIlink}$ graph, after $\text{constrCRlink}$ graph, after $\text{NetMelt}$ graph and after $\text{NetMelt}^+$ graph, respectively. GCCs of the graph after $\text{CRIlink}$ (3) and the graph after $\text{constrCRlink}$ (4) are similar to the original FB-1 graph’s GCC (1).

Figure 5: Comparison of the infected population under the SIS model. Our methods, $\text{CRlink}$ and $\text{constrCRlink}$, have similar infected rates in the population to $\text{NetMelt}^+$. But, our methods maintain the community structure of the population while $\text{NetMelt}^+$ does not.

Figure 6: The impact of parameter $\rho$ in $\text{constrCRlink}$, $\text{constrCRlink}$ with smaller $\rho$ leads to smaller $V(c, \hat{c})$ (i.e., less change in the community structure).
Figure 7: 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: Relationship between the average eigenscore value and edge degree \( d_e \) (sum of the two endpoints’ degrees). Larger degree edges tend to have higher eigenscores.

According to Figure 3, GRlink has the poorest performance in \( V(c, \hat{e}) \). 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 \( \varepsilon \varepsilon \) edges with both high degree endpoints and select low degree des nodes in edge rewiring. This is reasonable because most of the highest eigenscore \( \varepsilon \varepsilon \) edges have relatively high degree endpoints and most of the lowest eigenscore \( \varepsilon \varepsilon \) edges should connect relatively low degree nodes, as reported in Figure 8. With these finding, we can assume a common comparison between GRlink and CRlink, as illustrated in Figure 10.

From Figure 10, GRlink chooses \( \varepsilon \varepsilon \) 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 causes of the poor performance of GRlink and the good performance of CRlink:

- src and end nodes in CRlink hold stronger connection than that in GRlink. As shown by Figure 10, des node becomes common neighbor of src and end nodes in CRlink while des node is the only neighbor of src node in GRlink.

- In GRlink, due to low degree and weak connection with intra-community nodes, des nodes are easily reassigned to the community of src nodes. As shown by Figure 10, des nodes are in the 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,

\[
RCI_0 = \sum_{e_{ij} \in \Phi} \frac{I[\hat{e}(i) = e(j)] \cdot I[\hat{e}(i) = \hat{e}(j)]}{I[e(i) = e(j)]}
\]

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

For the second point,

\[
RCI_1 = \sum_{e_{ij} \in \Psi} \frac{I[e(i) \neq e(j)] \cdot I[\hat{e}(i) = \hat{e}(j)]}{I[e(i) \neq e(j)]}
\]

According to above two equations, \( RCI_0 \) quantifies the similarity of community assignments of \( \varepsilon \varepsilon \) edges’ two endpoints (i.e., src and end nodes) between the original graph and the perturbed graph, which should be measured in both GRlink and CRlink. \( RCI_1 \) quantifies the community reassignments of des nodes in GRlink. Figure 9 (a) and Figure 9 (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 maintaining community structure. GRlink performs badly in keeping community assignments of src and end nodes, and community reassign-
section, we measure the degree patterns of three kinds of nodes in keeping the same community assignment of src and end nodes. GRlink tends to reassign community of des node (high RCI) and constrCRlink performs well in keeping community assignment of des node (low RCI).

4.3.3 CRlink vs. constrCRlink

Results in Section 4.2.1 illustrate that constrCRlink performs best in V(c, ě), which means that we further improve the performance of CRlink by introducing node degree constraint. Similar to the analysis of GRlink and CRlink done in the previous section, we measure the degree patterns of three kinds of nodes in constrCRlink. Figure 7(c) reports that constrCRlink also chooses ε ≤ edges with high degree endpoints and selects within community des nodes with degree constraint (ρ = 1). In the same way, Figure 11 shows a case comparison between CRlink and constrCRlink.

According to Figure 11, 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.

- src and end nodes keep strong connection in constrCRlink. As illustrated in Figure 11, des nodes become common neighbors of 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.

Similarly, to quantify influence, we measure the RCI0 and RCI2 of constrCRlink, where RCI2 is defined as:

\[
RCI_2 = \sum_{i,j \in \Psi} \frac{I[\hat{c}(i) = c(j)] \cdot I[\hat{c}(i) \neq \hat{c}(j)]}{I[c(i) = c(j)]}
\]

Obviously, RCI2 quantifies the ability of constrCRlink in preventing community reassignments of des nodes. From the plot in Figure 9(a), we observe that RCI0 of constrCRlink is above 0.5 at most budgets and higher than that of CRlink. Very low values (less than 0.12) of RCI2 in Figure 9(c) demonstrate the strong ability of constrCRlink in keeping des nodes in the same community. Both of these lead to the good performance of constrCRlink.

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 mini-
mum 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 how to minimize dissemination in a population (that is represented as a complex network) while maintaining its community structure (where community is defined as a group of people with more links between them than to outside members). Due to the poor performance of edge deletion 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 tries to improve 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 entity dissemination minimization. Besides, CRlink and constrCRlink perform much better in preserving community structure (only 4.5% change) than other methods (with 13.6% change). Furthermore, we investigate the reasons that led to 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