Community Preserving Lossy Compression of Social Networks

Hossein Maserrat
School of Computing Science
Simon Fraser University
Burnaby, Canada
hmaserra@sfu.ca

Jian Pei
School of Computing Science
Simon Fraser University
Burnaby, Canada
jpei@sfu.ca

***Abstract*—Compression plays an important role in social network analysis from both practical and theoretical points of view. Although there are a few pioneering studies on social network compression, they mainly focus on lossless approaches. In this paper, we tackle the novel problem of community preserving lossy compression of social networks. The trade-off between space and information preserved in a lossy compression presents an interesting angle for social network analysis, and, at the same time, makes the problem very challenging. We propose a sequence graph compression approach, discuss the design of objective functions towards community preservation, and present an interesting and practically effective greedy algorithm. Our experimental results on both real data sets and synthetic data sets demonstrate the promise of our method.**

***Keywords*-compression; social networks, communities.**

I. Introduction

Partly motivated by the recent success of many online social networking sites such as Facebook and Twitter, managing and analyzing huge social networks have attracted dramatic interest from both industry and academia. Many social networks are huge and ever growing, which present an essential challenge for social network analysis.

As illustrated in several recent studies [2], [3], [4], [17], compressing social networks can substantially facilitate mining and advanced analysis of huge social networks. Social network compression plays an important role in social network analysis from both practical and theoretical points of view. Practically, many advanced social network analysis tools are sensitive to the input size. Those methods are highly efficient when the data can be held completely or largely into main memory, but may become very costly on data mostly out of memory. If social networks can be compressed effectively and efficiently, it may help such advanced analysis tools to handle much larger social networks.

Compression is always achieved by utilizing some form of "regularity" in data. Thus, the compressibility of a social network can provide valuable insights into the structure of the network. For example, if a subnetwork can be compressed well, it may indicate that the members in the subnetwork share some regularity, or the subnetwork follows some structural patterns that are shared by similar subnetworks.

This research is supported in part by an NSERC Discovery Grant, a BCFRST NRAS Endowment Research Team Program project, and a GRAND NCE project. All opinions, findings, conclusions and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

As reviewed in Section II, almost all the existing social network compression methods target at lossless compression. We argue that lossy compression in fact is very interesting and useful for social networks, because it is well-known that large social networks are often noisy.

From the practical point of view, noise edges and vertices in large social networks may damp the quality of social network analysis. An appropriate lossy compression of a social network can discard the noise edges and vertices in the network. Consequently, the lossy compression may be present as the input of higher quality for social network analysis. In other words, lossy compression of social networks can serve as a preprocessing step in social network analysis. This goes far beyond just space saving.

From the theoretical point of view, lossy compression of social networks can help to discover the importance of edges and vertices in a social network, and identify noise edges and vertices. Immagine ideally we have a lossy compression method that preserves the important information about a social network and filters out noise. If the method can assign to each element in a social network (e.g., each edge and vertex) a priority of being included into a lossy compression, then the priority can be regarded as a good indicator of the importance of the element. The lower the priority of an element, the more likely the element is noise.

Lossy compression of social networks is interesting and important for social network analysis. At the same time, it is a very challenging problem. To achieve good lossy compression of social networks, we have to develop good methodology that can detect and preserve important information in social networks.

In this paper, we tackle the novel problem of community preserving lossy compression of social networks, and make several important contributions. First, we advocate community preserving lossy compression of social networks due to the importance of communities in social networks. To the best of our knowledge, we are the first to identify and tackle the problem. Second, we propose a sequence graph compression approach. We design a simple yet meaningful objective function that opts for community structure preservation. A heuristic algorithm is developed. Last, we report an empirical study on both synthetic and real data sets, which verifies the effectiveness of our method.

The rest of the paper is organized as follows. We review the related work in Section II. We describe the essential idea of graph compression using sequence graphs in Section III, and design the community preserving objective function in Section IV. We present the compression algorithm in Section V, and report the experimental results in Section VI. Section VII concludes the paper.

II. Related Work

Community finding and analysis in social networks have been extensively explored from multiple disciplines, such as computer science, physics, and sociology. Fortunato [9] presented a comprehensive survey on community detection in complex networks. In general, given a network, we want to compute a partitioning of the nodes to communities, where a community is a set of nodes such that edges inside the community are more likely than those going outside. Having this notion of community in mind, Lancichinetti *et al.* [14] introduced the LFR benchmark, which is a model that can produce random networks with implanted communities of variable size, while the degrees of the nodes follow a power law distribution. Fortunato [9] also discussed comparing two different community structures, a nontrivial question. In particular the Normalized Mutual Information (NMI) [6] is an information theory based measure designed to capture the similarity of two different partitions.

There are many community finding methods. For example, Girvan and Newman [11] gave an algorithm based on the concept of "betweenness". Newman and Girvan [19] and Clauset *et al.* [5] developed the notion of modularity. Good *et al.* [12] studied modularity landscape. Effective methods based on random walk [24], hierarchical optimization of modularity function [1] were developed recently.

A large body of work on assessing the quality of individual communities in real life social networks has been developed. *Conductance* [13] is a widely accepted measure for this purpose. Recently, Leskovec *et al.* [15], [16] studied *Network Community Profile* plot (NCP) of social networks. They pointed out that in real life social networks, there are small communities with low conductance. As the size increases, the communities, however, start to "blend in" with the rest of the network and become less community like.

Due to practical demands, lossless compression of social networks/web graphs attracted much attention lately. Boldi and Vigna [3] showed in particular web graphs are compressible down to almost two bits per edge. Chierichetti *et al.* [4] extended the framework [3] using shingle ordering instead of lexicographical ordering of web pages, in order to tackle other types of real world networks. Chierichetti *et al.* [4], however, suggested that the compressibility of web graphs is an exception among other types of real life networks. Exploiting an ordering of nodes, which captures the "regularity" of the network, is critical and challenging. Very recently, Vigna *et al.* [2] introduced a *layered label propagation* algorithm for reordering very large graphs. They showed that their method can improve the compression rates for web graphs and social networks. It is worth mentioning that the layered label propagation algorithm in essence is a clustering method built up on [22], [23]. In our previous study [17], we used the notion of multi-position linearization to compress real life networks. Multi-position linearization is a sequence of nodes, in which any node can appear multiple times. The intuition is that, in real life networks, a node can be part of several clusters. Thus, such a sequence can capture the cluster structures better. Then, using a constant size vector of bits the "local" connections of nodes can be encoded. Finding an optimal linearization, however, is not trivial.

Navlakha *et al.* [18] proposed a graph summarization scheme with an error bound that compresses social networks by aggregating the nodes in supernodes and replacing all edges between two groups of nodes by a superedge. They also keep a set of corrections in order to be able to recreate the original graph. Their approach allows lossy compression. Given an error rate ϵ, the objective is to reduce the size of representation. One critical difference between their method and our approach is that we assume the size of representation is given and our objective is to capture the community structure of a network.

Most recently, Fan *et al.* [8] suggested a query preserving graph compression framework. Their approach does not store the original graph, rather, it computes the equivalence classes of nodes according to a given class of queries. Then, it builds a smaller graph that has the equivalence classes as the vertices. This approach is quite effective for simple queries (e.g. reachability) and less effective for more complex queries (e.g. pattern matching). Their method does not target at community preservation.

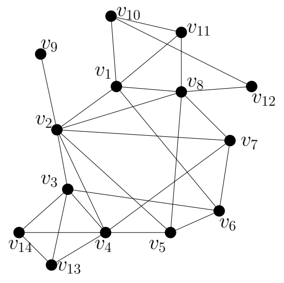
III. Graph Linearization for Lossy Compression

In our previous study [17], we developed the notion of graph linearization for lossless graph compression. In this section, we extend the notion of graph linearization to allow lossy compression.

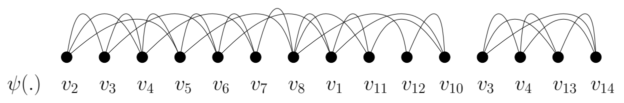
For the sake of simplicity, in this paper, we model a social network as an *undirected simple graph* $G = (V, E)$, where V is a set of vertices, $E \subset V \times V$ is a set of edges, and $(u, u) \notin E$ for any $u \in V$. We also refer to V by $V(G)$ and to E by $E(G)$. Our discussion can be straightforwardly extended to directed and non-simple graphs.

We first formulate a notion of sequence graph.

Definition 1 (Sequence graph)**.** *A graph G_s is a (k, l)-**sequence graph**, if $|V(G_s)| = l$ and there is a bijection ϕ between $V(G_s)$ and the set of integers $\{1, \ldots, l\}$ such that for every edge $(x, y) \in E(G_s)$, $|\phi(x) - \phi(y)| \leq k$.*



(a) Graph G



$\psi(.)$
v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_1 v_{11} v_{12} v_{10} v_3 v_4 v_{13} v_{14}

(b) A $(3, 15)$-sequence graph G_s that is a lossy linearization of G.

Figure 1. A graph G and its lossy representation using a $(3, 15)$-sequence graph G_s.

We call k the ***local range size****, l the* ***sequence length****, and $span(x, y) = |\phi(x) - \phi(y)|$ the* ***span*** *of edge (x, y).*

Intuitively, in a sequence graph, the vertices can be lined up into a sequence so that all edges are "local", that is, the two end points locate within a segment of at most k in the sequence. Since ϕ is a bijection between $V(G_s)$ and integers $\{1, \ldots, l\}$, hereafter, we may simply refer to the vertices in G_s by integers in $\{1, \ldots, l\}$, and may draw the vertices of a sequence graph in a sequence and omit the integers if they are clear from the context.

Example 1 (Sequence graph)**.** *Graph G_s in Figure 1(b) is a $(3, 15)$-sequence graph. Please note that we simply line up the vertices in a sequence and omit the integers in the graph. $\psi(\cdot)$ in the figure is for Example 2 and should be ignored at this moment.*

In general, a (k, l)-sequence graph G_s may have more than one bijection between $V(G_s)$ and integers $\{1, \ldots, l\}$. Our discussion applies to all bijections unless specifically mentioning.

To store a (k, l)-sequence graph, for each vertex, we only need to allocate $2k$ bits to represent the edges involving the vertex. This representation can also enable efficient neighborhood queries — finding all neighbors of a vertex u takes only $O(k)$ time.

The general idea behind graph compression using linearization is that we try to "unfold" a graph into a sequence graph, so that many vertices have the associated edges in their local ranges. Then, storing the corresponding sequence graph can save space, because many edges are stored using only 2 bits each, one for each end point. We refer to this process as "unfolding" because a vertex in the original graph may be mapped to several vertices in the sequence graph.

Definition 2 (Graph linearization)**.** *A (k, l)-sequence graph G_s is a (k, l)-**linearization** of a graph G if there exists a function $\psi : V(G_s) \rightarrow V(G)$ such that (1) for every edge $(x, y) \in E(G_s)$, $(\psi(x), \psi(y)) \in E(G)$, and (2) there do not exist two edges $(x, y), (x', y') \in E(G_s)$, $(x, y) \neq (x', y')$ such that $(\psi(x), \psi(y)) = (\psi(x'), \psi(y'))$. To keep our notation simple we overload the symbol ψ by writing $\psi(x, y) = (\psi(x), \psi(y))$.*

G_s is a ***lossless linearization*** *[17] of G if for every edge $(u, v) \in E(G)$, there exists an edge $(x, y) \in E(G_s)$ such that $\psi(x, y) = (u, v)$. Otherwise, G_s is a* ***lossy linearization*** *of G.*

The second condition in Definition 2 ensures that an edge in the original graph is encoded at most once in the linearization. This condition helps us to design a simple yet effective objective function for lossy compression in the next section.

Example 2 (Lossy linearization)**.** *The $(3, 15)$-sequence graph G_s in Figure 1(b) is a lossy linearization of graph G in Figure 1(a). The mapping $\psi(\cdot)$ from the nodes of G_s to the nodes of G is depicted.*

The problem of finding a (k, l)-lossless linearization of G that minimizes l is also known as computing MP$_k$-linearization of graphs [17]. We [17] showed that MP$_k$-linearization is a very challenging problem in general, though an optimal algorithm exists for $k = 1$.

In general, a graph G may have multiple (k, l)-lossy linearizations. Finding the best (k, l)-lossy linearization for a graph G is a novel problem not touched by any previous work. To make the problem concrete, we need to explore how to quantify the "loss of information" and assess the degree of community preservation in lossy compression. We answer this question next by designing an objective function.

IV. Objective Function Design

Let us consider the following optimization problem. Given a graph G and parameters $l > 0$ and $k > 0$, find a (k, l)-lossy linearization G_s for G and the mapping $\psi : V(G_s) \rightarrow V(G)$ such that a utility objective function $f(G_s)$ is maximized, where $f(G_s)$ measures the goodness of G_s in preserving the information in G.

Lossy compression trades off some edges in the original graph G for space saving. What information in G should be preserved in priority? Since communities are the essential building blocks of social networks, in this paper, we focus on lossy compressions of social networks that preserve communities. We regard a dense area in a graph as a potential community, and intently avoid an exact definition of community, since different applications may have different definitions.

We want to obtain a utility function that opts for edges of short spans in the corresponding sequence graph. Instead of developing a utility function parameterized by local range

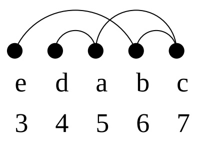


Figure 2. The span of a path.

size k, we introduce a parameter α $(0 < \alpha < 1)$ that controls the strength of preference on shorter spans. We will build the connection between parameters α and k in Section V-B.

A *path* $p = (u_1, u_2, \ldots, u_m)$ in a graph G is a series of edges such that $(u_i, u_{i+1}) \in E(G)$, $1 \le i < m$. The *length* of path p is $(m-1)$, the number of edges involved in the path. In a linearization G_s of G under mapping ψ, path $p' = (u'_1, u'_2, \ldots, u'_m)$ in G_s is the *embedding* of path p if $\psi(u_i, u_{i+1}) = (u'_i, u'_{i+1})$ for $1 \le i < m$.

Definition 3 (Span of path)**.** *Let G_s be a linearization of graph G, $p = (u_1, u_2, \ldots, u_m)$ a path in G, and $p' = (u'_1, u'_2, \ldots, u'_m)$ the embedding of p in G_s. The* ***span*** *of p is*

$$span(p) = \max_{1 \le i \le m} \{\phi(u'_i)\} - \min_{1 \le i \le m} \{\phi(u'_i)\}$$

Example 3. *Figure 2 shows a segment of a $(3, l)$-sequence graph. For path $p = (d, a, c, b, e)$, the span is $7 - 3 = 4$.*

Let us start our design of the objective function by considering a simple function. Suppose G_s is a (k, l)-linearization of G, where $k = l = |V(G)|$. If we only consider individual edges, and try to shorten the sum of spans of all edges, then we can use the following utility function

$$f_1(G_s) = \sum_{(x,y) \in E(G_s)} \alpha^{span(x,y)}$$

Utility function f_1 has the following two properties.

Property 1: the shorter the spans of edges in the sequence graph, the higher the utility. This property is consistent with our goal of preserving community information. A community typically has a high density, which means there exist many edges among the set of vertices belonging to the community. If the vertices of a community are placed in proximate positions in the sequence, the spans of the edges within the community tend to be short. The edges of long spans contribute little to the utility. The utility decreases exponentially with respect to the span. This property encourages the arrangement of vertices belonging to the same community in the close-by positions, and discourages the inclusion of edges crossing communities far away in the original graph G.

Property 2: the more edges included in the compression, the higher the utility. Consider two linearization graphs G_s and G'_s such that $V(G_s) = V(G'_s)$ and $E(G_s) \subset E(G'_s)$. Then, $f_1(G_s) < f_1(G'_s)$. This property encourages a linearization graph to include as many edges as possible in addition to optimizing for short span edges.

Utility function f_1 is sensitive to individual edges. We can extend it to incorporate community information better. Instead of edges, we can consider how paths of a certain length are represented in a sequence graph. Generally, a community as a dense subgraph has many short paths traversing among members within the community. If a sequence graph preserves the community information, then the members of the community are lined up close to one another in the sequence graph and thus the paths in the community fall into short ranges of the sequence.

To incorporate the above idea, let $P_m(G_s)$ be the set of paths of length m in a sequence graph G_s. We can extend utility function f_1 to

$$f_m(G_s) = \sum_{p \in P_m(G_s)} \alpha^{span(p)}$$

Clearly, utility function f_m is a generalization of f_1. The longer the paths are considered, the more community oriented the utility function becomes. At the same time, the optimization problem becomes more challenging when the value of m increases.

Observe that function f_1 takes its maximum value when the span of each edge is one, and that is basically an adjacency representation of the graph. In this paper, we focus on the simplest nontrivial setting $m = 2$ as the first step. Interestingly, several recent studies, such as [10], suggested that even considering random walks of short length can generate high quality results in network analysis. Note that for $m \ge 3$, the problem is computationally more expensive. Optimizing f_m for larger values of m is the subject of future studies.

For the sake of simplicity, we omit the subscript 2 hereafter, and tackle the optimization of the following objective function:

$$f(G_s) = f_2(G_s) = \sum_{p \in P_2(G_s)} \alpha^{span(p)} \qquad (1)$$

V. Linearization Method

In this section, we derive upper and lower bounds of the objective function, build the connection between parameters α and k, and develop a greedy heuristic linearization method.

A. Bounding the Objective Function

How difficult is the problem of finding the optimal lossy linearization using utility function f in Equation 1, that is, finding a sequence graph maximizing the objective function? In literature, there is a family of *graph layout problems* [7], whose objective is to find an *ordering* of nodes to optimize a particular objective function. Many variants of these problems have been shown to be NP-hard [21], [4]. To the best of our knowledge, even no constant factor approximation algorithm for any variation of these problems is known [25], [7]. Note that our setting is even more complex, since one node can appear in several positions in a sequence graph.

These evidences suggest that very likely the problem is not solvable in polynomial time, unless $P = NP$. Therefore, in this section, we design a greedy heuristic method.

In order to obtain effective greedy heuristics, we try to bound the objective function. We observe the following.

Theorem 1 (Bounds). *Let G_s be a sequence graph. Then,*

$$f(G_s) \leq \sum_{p \in P_2(G_s)} (\alpha^{1/2})^{span(e_1)+span(e_2)} \quad (2)$$

$$f(G_s) \geq \sum_{p \in P_2(G_s)} \alpha^{span(e_1)+span(e_2)} \quad (3)$$

Proof: For any path $p = e_1 e_2$ in G_s, we have $span(p) \geq \max\{span(e_1), span(e_2)\} \geq \frac{span(e_1)+span(e_2)}{2}$. Since $0 < \alpha < 1$ Equation 2 follows immediately.

Apparently, $span(p) \leq span(e_1) + span(e_2)$. Thus, Equation 3 holds.

In Equations 3 and 2, $\alpha^{\frac{1}{2}}$ and α, respectively, are constants. Heuristically, if we can obtain a sequence graph optimizing the lower bound in Theorem 1, the sequence graph may have a good chance to boost the objective function f.

Let E_i be the set of edges incident to vertex i in G_s and P_i the set of those paths of length two that have vertex i as the middle vertex. Then,

$$(\sum_{e \in E_i} \alpha^{span(e)})^2 = \sum_{p=e_1e_2 \in P_i} \alpha^{span(e_1)+span(e_2)}$$

Therefore, we optimize the lower bound in Theorem 1 if we optimize the following nice double summation:

$$\bar{f}(G_s) = \sum_{1 \leq i \leq |V(G_s)|} (\sum_{e \in E_i} \alpha^{span(e)})^2$$

B. Connection between Parameters α and k

Given α, we have the following interesting upper bound on the span of any edge in the optimal sequence graph.

Theorem 2. *For a given α, the maximum span of all edges in the optimal sequence graph is at most $\log_\alpha \frac{\alpha(1-\alpha)}{4}$.*

Proof: Let $w_i = \sum_{e \in E_i} \alpha^{span(e)}$, where E_i is the set of edges associated with position i.

$$w_i = \sum_{e \in E_i} \alpha^{span(e)} < 2 \sum_{i=1}^{\infty} \alpha^i$$

$\sum_{i=1}^{\infty} \alpha^i$, however, is the sum of a geometric sequence and is equal to $\alpha/(1-\alpha)$. Thus we can rewrite the inequality in the following form.

$$w_i < \frac{2\alpha}{1-\alpha}$$

The contribution of edge e is at most

$$\begin{aligned} w_i^2 - (w_i - \alpha^{span(e)})^2 &= 2\alpha^{span(e)} w_i - \alpha^{2span(e)} \\ &< \frac{4\alpha^{span(e)+1}}{1-\alpha} - \alpha^{2span(e)} \end{aligned}$$

This value should be larger than the contribution of a single isolated edge, otherwise removing this edge and adding it to the end of the sequence graph would increase the objective function. Thus,

$$\alpha^2 < \frac{4\alpha^{span(e)+1}}{1-\alpha} - \alpha^{2span(e)}$$

$$\frac{\alpha(1-\alpha)}{4} < \frac{\alpha^2(1-\alpha)}{4\alpha - \alpha^{span(e)}(1-\alpha)} < \alpha^{span(e)}$$

Since $0 < \alpha < 1$ and $span(e)$ is an integer, we have

$$span(e) < \log_\alpha \frac{\alpha(1-\alpha)}{4}$$

Our problem formulation assumes a parameter k is given as the maximum local range size for the sequence graph. The objective function, however, uses parameter α. Theorem 2 builds the analytical ground to connect α and k. We use the equation $k = \log_\alpha \frac{\alpha(1-\alpha)}{4}$ to estimate α. Specifically, to estimate α given k, we do a binary search on the interval $[0, 1]$, and stop when the value of $\log_\alpha \frac{\alpha(1-\alpha)}{4}$ is between k and $k - 0.01$. The binary search is effective because the function $\log_\alpha \frac{\alpha(1-\alpha)}{4}$ is monotonically increasing in the interval $[0, 1]$. Using this estimate of α, experimentally we observe that in the resulting sequence graphs the spans of an extremely small fraction of edges are more than $k/2$. This is consistent with Theorem 2. Therefore, to not waste memory, we use $2k = \log_\alpha (1-\alpha)\alpha/4$ to estimate α.

C. A Greedy Heuristic Method

In this section, we develop a greedy heuristic method for the community preserving lossy compression problem. We will use a local search heuristic.

1) Overview and General Ideas: The basic operation for local improvement in our heuristic is that, given a node, we find a position in the sequence to insert a new copy of the node, and find a position to delete such that the total change in the objective function is positive after the insertion and deletion. Similar to most local search heuristic methods, our method does not have any theoretical guarantee for the convergence time or the quality of the result. Using an extensive set of experiments, as reported in Section VI, we verify the effectiveness of our design in practice.

Algorithm 1 shows our main algorithm. we initialize G_s with a random ordering of the vertices of G (Line 1). There is no edge in G_s at this stage. Then, iteratively we consider all vertices for possible reallocation. The $ReAllocate(u, G, G_s, \alpha)$ procedure (Algorithm 2) returns a position in G_s for possible insertion of an extra copy of u and its associated edges. If the length of G_s is already l, the algorithm searches the local range of the insertion point for a possible deletion. We apply the changes if they improve the objective function.

To implement this algorithm we need a data structure to store the sequence graph G_s, which allows fast insertion and deletion operations. We explain our data structure next.

Algorithm 1 Compression Algorithm

```
Require: $G$: input network, $k$: local range,
    $l$: length of compression ($l \geq |V(G)|$)
Ensure: $SeqG$: sequenced compression
 1: Initialize $SeqG$ with a random ordering of nodes
 2: $\alpha \leftarrow EstimateAlpha(k)$
 3: repeat
 4:   $b \leftarrow f(SeqG, \alpha)$
 5:   for all $u \in V(G)$ do
 6:     $IPos \leftarrow NULL$, $DPos \leftarrow NULL$
 7:     $(IPos, Nbh) \leftarrow ReAllocate(u, G, SeqG, \alpha)$
 8:     if $(IPos \neq NULL)$ and $(Length(SeqG) = l)$ then
 9:       $DPos \leftarrow SeqG.LowestBenf(IPos - k, IPos + k)$
10:     end if
11:     $x \leftarrow UtilityIncrease(IPos, Nbh, SeqG)$
12:     $y \leftarrow UtilityDecrease(DPos, SeqG)$
13:     if $x - y > 0$ then
14:       $Insert(IPos, Nbh, SeqG)$
15:       $Delete(DPos, SeqG)$
16:     end if
17:   end for
18:   $\alpha \leftarrow f(SeqG, \alpha)$
19: until convergence condition
```

2) SeqGraph Data Structure: Similar to the Eulerian data structure we need to store a sequence of cells (Figures 3(a) and 3(b)), where each cell represents a position in G_s. Each cell contains two pieces of information: a *next-copy pointer* to the next copy of the same vertex, and a vector of $2k$ bits to represent the local edges. All copies of the same vertex form a cyclic linked list, which is referred to it as a *vertex cycle*. The Eulerian data structure [17] uses an array, in which the cost of inserting and deleting a cell is linear.

In our heuristic algorithm, we have to frequently insert and delete cells. Even the linear cost in insertion and deletion is too expensive. Thus, we need a better data structure to avoid the cost of shifting long segments in the sequence graph in insertions and deletions. Sepcifically, we divide the cells into segments. Each segment has up to M cells and is stored in an array. Then, the SeqGraph data structure is a double linked list of segments. In Figure 3(c), a, b and c are the segments. In each cell, a next-copy pointer is stored. For example, in the first cell of segment a, the next-copy pointer b:4 points to the fourth cell in segment b. To point to a cell, unlike the Eulerian data structure [17] where an integer index can be simply used, we need to use an index consisting of a pointer to a segment and an offset in that segment.

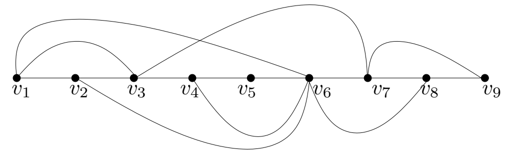
Fortunately, we only need to search within the range size k. That is, in Algorithm 2, we only need to compute the exact distance between positions i_1 and i_2 if the distance is up to k. This can be achieved efficiently by searching a small neighborhood.

Algorithm 2 Reallocation Procedure

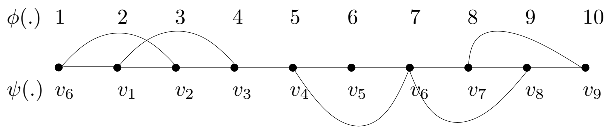
```
Require: $G$: original graph, $SeqG$: sequence graph, $u \in V(G)$, $\alpha$: weighting parameter
Ensure: $IPos$: potential position to insert a new copy of $u$, $Nbh$: the edges associated to the new copy of $u$
 1: $C_1 \leftarrow \psi^{-1}(u)$; $C_2 \leftarrow \emptyset$;
 2: for all $\{v | (u, v) \in E(G \setminus SeqG)\}$ do
 3:   Let $p$ be a random member of $\psi^{-1}(v)$;
 4:   $C_2 \leftarrow C_2 \cup \{p\}$;
 5:   $N(p) \leftarrow \{+1\}$; /* neighbors of $p$ in $SeqG$ */
 6:   $U(p) \leftarrow \alpha$; /* utility of $p$ */
 7:   $SeqG.Insert(u, p)$;
 8: end for
 9: for all $p \in C_1$ do
10:   $N(p) \leftarrow SeqG.Nbh(p)$; $U(p) \leftarrow \sum_{a \in N(p)} \alpha^{|a|}$;
11: end for
12: $C \leftarrow C_1 \cup C_2$;
13: repeat
14:   for all $\{v | (u, v) \in E(G)\}$ do
15:     Let $p_u$ and $p_v$ be s.t. $(\psi(p_1), \psi(p_2)) = (u, v)$;
16:     /* $p_u \in C$ and $p_v \in \psi^{-1}(v)$ */
17:     $a \leftarrow Dist(p_u, p_v)$; /* the distance in $SeqG$ */
18:     $U(p_u) \leftarrow U(p_u) - \alpha^{|a|}$;
19:     Let $p_u^* \in C$ and $p_v^* \in \psi^{-1}(v)$ be s.t. maximize:
            $(\alpha^{|dist(p_u^*, p_v^*)|} + U(p_u^*))^2 - U(p_u^*)^2$
20:     $U(p_u^*) \leftarrow U(p_u^*) + \alpha^{dist(p_u^*, p_v^*)}$;
21:     if $p_u^* \neq p_u$ then
22:       $N(p_u^*) \leftarrow N(p_u^*) + \{dist(p_u^*, p_v^*)\}$;
23:       $N(p_u) \leftarrow N(p_u) - \{dist(p_u, p_v)\}$;
24:     end if
25:   end for
26: until Convergence
27: for all $p \in C_1$ do
28:   $SeqG.UpdateNeighbor(p, N(p))$;
29: end for
30: for all $p \in C_2$ do
31:   $SeqG.Delete(p)$;
32: end for
33: Let $p \in C_2$ s.t. $U(p)$ is maximum;
34: $IPos \leftarrow p$; $Nbh \leftarrow N(p))$;
35: return $(IPos, Nbh)$;
```

Let M be the maximum number of cells in a segment. Without loss of generality, we assume M an even number. If an insertion operation causes a segment to have M cells, we split the segment into two segments of equal size. Moreover, if a deletion operation results in the sum of the lengths of two consecutive segments equal to $M/2$, we merge them. This is to avoid having many tiny segments in the data structure.

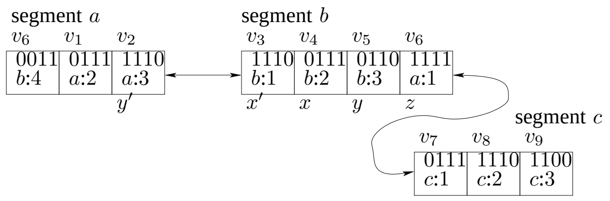
For an insertion (deletion) operation, a shift in the affected



(a) Graph G


$\phi(.)$ 1 2 3 4 5 6 7 8 9 10
$\psi(.)$ v_6 v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9

(b) $(2,10)$-Sequence Graph G_s and the mapping ψ to original vertices of G



(c) SeqGraph data structure (the cell ids in cells, such as $b:4$, are the next-copy pointers to the next copy of the same vertices)

Figure 3. A graph G and its representation by SeqGraph data structure.

segment is necessary. Moreover, the next pointers should be updated for those that point to the positions affected by the insertion and deletion, that is, the position of insertion or deletion and the positions thereafter in the same segment. A nice property of our SeqGraph data structure is that all other segments are not affected. Finally, those edges that pass over the insertion (deletion) position should be updated, too.

Example 4. *In Figure 3(c), let us call the cells b:2, b:3 and b:4, respectively, by x, y and z, as labeled in the figure. To insert a copy of v_7 at position b:2, a shift in the segment b is needed. Moreover, we have to change the next-copy pointers of all the cells that point to x, y and z to b:3, b:4 and b:5, respectively. Note that we can find these pointers by following the vertex cycles of x, y and z, respectively. The edges associated with those cells that are at most 2 positions away from the inserted cell should be updated, that is x, x', y and y'. Finally, the newly inserted cell should be added to the vertex cycle of v_7.*

The cost of the insertion and deletion operations depends on the sizes of segments, vertex cycles and maximum span of edges. The time complexity of an insertion or deletion operation is $O(M\Delta + k^2)$, where Δ is the maximum degree of the graph.

3) The Reallocation Procedure: For a node u, the reallocation procedure (Algorithm 2) partitions the edges incident to u into groups. Each group is associated with a copy of u in G_s. Let C_1 be the set of positions of copies of u that already exist in G_s, and C_2 the set of positions of potential new copies. C_2 is generated as follows: let $E(G \setminus G_s)$ be the set of edges $(u, v) \in E(G)$ that are not represented in G_s. For $(u, v) \in E(G \setminus G_s)$, we add a potential new copy of u right behind a random copy of v in G_s (Lines 3-7).

We insert these new potential copies at the beginning, and delete them all at the end of the procedure (Lines 1-12). In the edge reassigning step (Lines 13-26), each edge is added to the group for which it has the best contribution (Line 19). The reassigning process stops when no improvement is possible. At the end, for the existing copies of u, that is, C_1, the associated edges will be updated if they have been changed (Lines 27-29). For those potential new copies, that is, C_2, we remove them and return the best in the set as the position of a potential new copy of u in G_s. The utility is also returned (Lines 30-35).

4) Running Time Analysis: Denote by d_v the degree of node v. We have at most d_v groups. Inserting and deleting the groups in the SeqGraph data structure takes $O(d_v(M\Delta + k^2))$ time. In an iteration of the edge reassigning process, all edges have to be considered for reassigning. Each edge takes $O(d_v k)$ time. In total each iteration takes $O(d_v^2 k)$ time.

We notice that the number of iteration in practice is very small. Although we do not have any theoretical bound on the number of iterations, since the contribution of any single edge is increasing, the convergence is guaranteed.

Likewise, the overall time complexity of Algorithm 1 depends on the convergence condition and the number of iterations. At the same time, the complexity of a single iteration can be estimated as follows. Assuming the number of iterations for the reallocation process is at most I, the running time is proportional to $\sum_{v\in V(G)} d_v(d_v kI + M\Delta + k^2)$. Since k and M are constants, the sum is proportional to $\sum_{v\in V(G)} d_v(d_v I + \Delta)$. Notice that d_v is at most Δ, therefore, the time complexity of an iteration is $O(|E|\Delta I)$.

VI. Empirical Evaluation

We report a systematic empirical evaluation.

A. Evaluation Methodology

Our compression algorithm assumes the parameters k, the local range size, and l, the length of sequenced compression. Having these parameters fixed, the size of compression can be computed precisely. Therefore, compression rate does not have a straightforward meaning here. Instead, we consider a measure to assess the utility of a single bit.

Definition 4 (Bit-utility rate)**.** *The **bit-utility rate** is the ratio of the number of edges encoded in the lossy compression over the total number of bits.*

To evaluate the quality of a lossy compression, one has to look at the quality of community preservation. This is a challenging task. It is hard to find a ground truth for

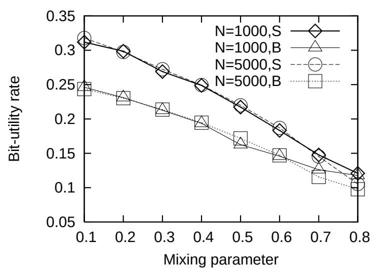


Figure 4. The bit-utility rate of our lossy compression.

the community structure of real world networks. Moreover, our method does not even explicitly identify communities in social networks.

For a sanity check of our method, we design the following methodology. Using the distance of the vertices in the sequence graph compression, we define a *proximity graph* (Definition 5), which is a weighted graph on the vertex set of the original network. The weight of edge (u, v) is an indicator for u and v belonging to the same community. Then, we run a simple community structure detection algorithm on both the original network and the proximity graph. A significant improvement in community detection should be interpreted as the effectiveness of our community preservation lossy compression method. To have a ground truth, we use synthesis networks with implanted communities.

The second experiment is concerned with the effect of missing edges in the lossy compression. We compare three different centrality measures on both the network with the original set of edges and the network with the set of those edges that are encoded in the compressed version. We use a collection of real world networks in this experiment.

B. Synthesis Networks

The LFR [14] benchmark is a random model to generate networks with implanted communities of variable size, while the degree of the nodes follows a power law distribution. We use the algorithm by Clauset *et al.* [5] for community structure detection. This is a simple and scalable greedy modularity-based algorithm. As reported by Fortunato [9] in his Figure 33, this algorithm has a poor performance on the LFR benchmark. A significant improvement of the performance of this algorithm over our lossy representation of the network supports our claim that our method is more than a compression framework and can serve as a preprocessing step for clustering analysis.

To generate synthesis networks, we use the same settings as those used by Fortunato [9]. We set the parameters as follows: the average degree to 20, the maximum degree to 50, the exponent for degree distribution to 2, and the exponent for the community size distribution to 1.

In Figure 4, N is the number of nodes, S stands for the data sets where the community sizes are made between 10 and 50, while B stands for the data sets where the community sizes are set between 20 and 100. The X-axis is the mixing parameter, which is the fraction of edges going outside of the community for any particular vertex, used by the data generator [14]. The Y-axis is the bit-utility rate. A larger bit-utility rate means that the lossy compression scheme makes a better use of the available space. As expected, if the density of the communities decreases, the bit-utility rate decreases, too.

We define a proximity graph as follows.

Definition 5 (Proximity graph)**.** *Let G_s be a linearization of G, and $\psi : V(G_s) \rightarrow V(G)$ the mapping. Note that $V(G_s) = \{1, \cdots, |V(G_s)|\}$. The* ***proximity graph*** *of G with respect to G_s is defined as follows. Consider $(u, v) \in E(G)$ and $(i, j) \in E(G_s)$ such that $|i - j| \leq k$ and $\psi(i) = u$, $\psi(j) = v$. Without loss of generality we assume $i < j$. The weight of undirected edge (u, v) in the proximity graph is*

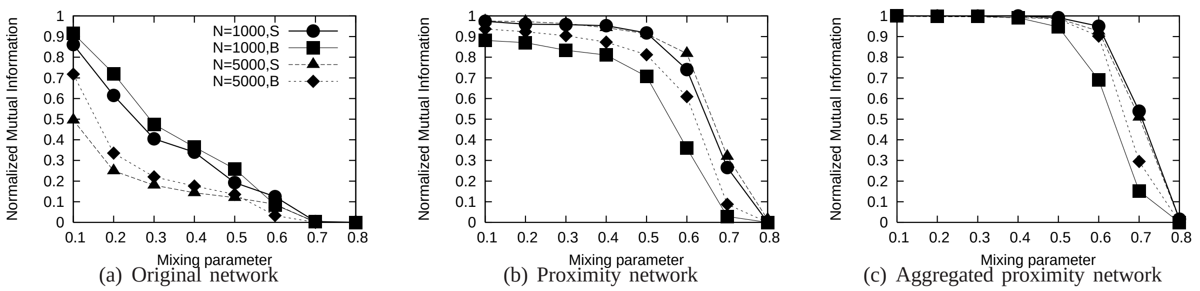
$$\sum_{(i,l)\in E(G_s),l\geq j} \alpha^{|i-l|} + \sum_{(j,l)\in E(G_s),l\leq i} \alpha^{|j-l|}$$

The first (second) term is over all the edges associated with position i (j) that pass over position j (i). If there is no such a pair of (i, j), then the weight for (u, v) is 0. If there are more than one such pair, for each of those pairs, we compute the weight and take the sum over all of them.

The merit of this weighting schema is that, when i and j belong to a dense part of the sequence graph G_s, $(\psi(i), \psi(j))$ receives a relatively large value, even if there is no direct edge between i and j. In practice, there is no need to explicitly store the proximity graph, rather one can compute the weights from the linearization graph on the fly. Note that, since the unfolding algorithm uses a random order of the vertices as the starting seed, to exploit the power of randomization, one can use several independent linearizations to derive an aggregate proximity graph, that is, the weight of (u, v) is the sum of its weights in the independent proximity graphs.

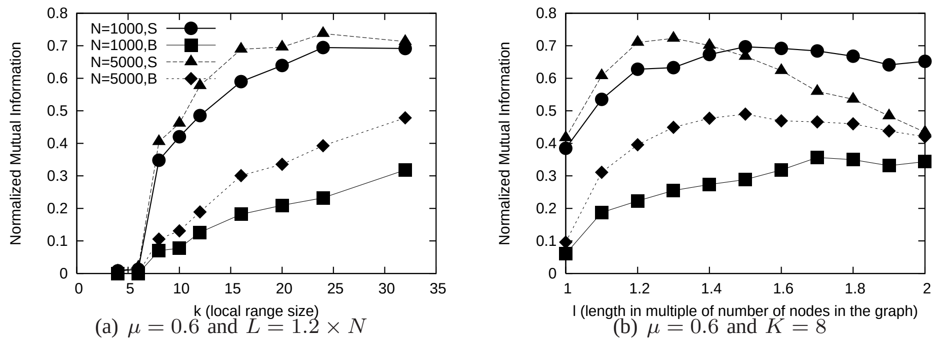
Figure 5 shows the results of the algorithm by Clauset *et al.* [5] on the original graph, the proximity graph and the aggregate proximity graph on five independent sequenced compressions. In those experiments, $k = 16$ and $l = 1.2\times N$, where N is the number of vertices in the original graph. The Y-axis is the Normalized Mutual Information (NMI) [6], [9], which measures the similarity between two clusterings, that is, community structures, of the same set of nodes. We compute NMI using the code at http://sites.google.com/site/andrealancichinetti/mutual.

The results clearly show that the algorithm is not effective on the original graph, but performs significantly better on the proximity graph. Moreover, the results on the aggregate proximity graph are comparable to the state-of-the-art algorithms reported by Fortunato [9] in his Figure 33. This strongly suggests that our method captures the community



(a) Original network (b) Proximity network (c) Aggregated proximity network

Figure 5. The performance of the algorithm by Clauset *et al.* [5] on the original network, the proximity network derived from one compression, and the aggregated proximity network derived from five independent compressions.



(a) $\mu = 0.6$ and $L = 1.2 \times N$ (b) $\mu = 0.6$ and $K = 8$

Figure 6. The effect of (a) local range size (k) and (b) length of sequence (l).

structure of the network, and can serve as an effective preprocessing step.

Figure 6 shows that our framework can benefit from increasing local range size k. The benefit of increasing the length of sequenced compression, however, is limited for the community detection quality due to the weakness of the detection algorithm [5].

C. Centrality

Betweenness [11] is a centrality measure of vertices for a given graph. It gives higher values to those vertices that occur on many shortest paths between other pairs of vertices. PageRank [20] is another centrality measure that uses random walks in a network to evaluate the importance of nodes. The degree of a vertex also can be considered as a simple centrality measure. In this section, we evaluate the effect of information loss in lossy compression on those centrality measures.

For a graph G and its sequence graph compression G_s, $\psi(G_s)$ is a graph on vertex set $V(G)$. Edge $(u, v) \in E(G)$ is an edge in $\psi(G_s)$ if and only if there exists $(i, j) \in E(G_s)$ such that $(u, v) = (\psi(i), \psi(j))$. Intuitively, $\psi(G_s)$ consists of those edges in G that are encoded in G_s.

Each vertex has a centrality score. A centrality measure on a network G can be regarded as a *centrality vector* of size $|V(G)|$. To make comparison we use the Pearson correlation of the centrality vectors of G and $\psi(G_s)$.

	Description	#nodes	#edges
ca-GrQc	Coll. net. of Arxiv Gen Relativity	5242	14990
em-ExCh	UVR Email Exchange	1133	5451
wiki-vote	Wikipedia who-votes-on-whom network	7115	100763

Table I
STATISTIC OF DATA-SETS

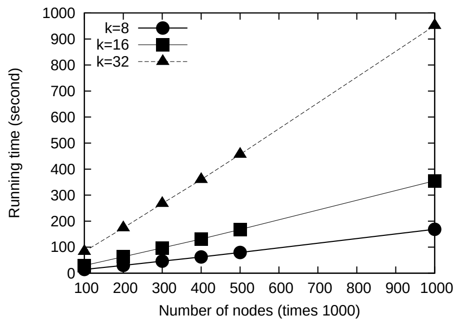
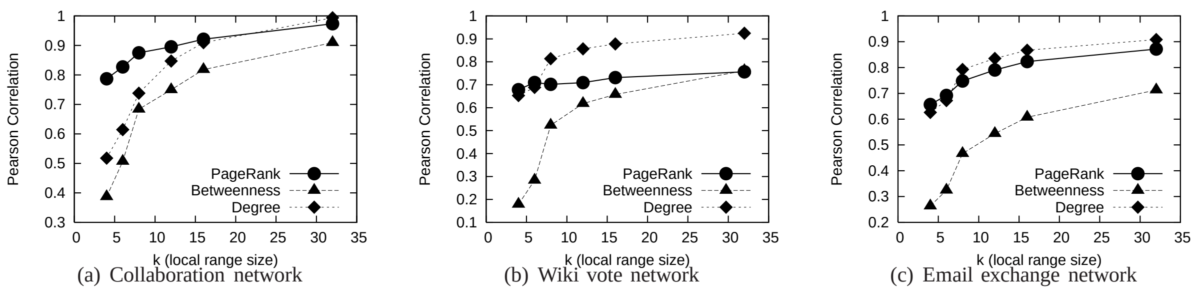


Figure 8. The running time of a single iteration of Algorithm 1.

Figure 7 shows the results on three real world networks whose statistics are given in Table I. As the value of k increases, the correlation between the two centrality vectors increases.



(a) Collaboration network (b) Wiki vote network (c) Email exchange network

Figure 7. Pearson Correlation of centrality measures on original and the compressed version

D. Running Time Efficiency

The running time of our method depends on the number of iterations in Algorithm 1. For our setting the number of iterations is typically between 15 to 30. Figure 8 depicts the running time of our method with respect to the graph size on the same LFR benchmarks [9]. The running time of a single iteration, when parameter k is fixed, is scaling almost linearly. Note that the result indicates that our method is efficient on networks up to millions of nodes.

VII. Conclusions

In this paper, we developed a lossy compression scheme for social networks. In our design, the quota size of compression is given, and the objective is to preserve the community structure of a given network as much as possible. We tackled this problem by introducing a notion of sequence graph. Moreover, we designed an objective function that measures the quality of a sequence graph with respect to the community structure of the network. We presented a non-trivial heuristic method to optimize our objective function. Finally, we validated our method using both synthesis and real world networks.

References

[1] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks, 2008.

[2] P. Boldi, M. Rosa, M. Santini, and S. Vigna. Layered label propagation: a multiresolution coordinate-free ordering for compressing social networks. In *WWW*, pages 587–596, New York, NY, USA, 2011. ACM.

[3] P. Boldi and S. Vigna. The webgraph framework I: compression techniques. In *WWW*, pages 595–602, 2004.

[4] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan. On compressing social networks. In *KDD*, pages 219–228, 2009.

[5] A. Clauset, M. E. J. Newman, , and C. Moore. Finding community structure in very large networks. *Physical Review E*, pages 1– 6, 2004.

[6] L. Danon, J. Duch, A. Diaz-Guilera, and A. Arenas. Comparing community structure identification, Oct. 2005.

[7] J. Díaz, J. Petit, and M. J. Serna. A survey of graph layout problems. *ACM Comput. Surv.*, 34(3):313–356, 2002.

[8] W. Fan, J. Li, X. Wang, and Y. Wu. Query preserving graph compression. In *SIGMOD*, pages 157–168, New York, NY, USA, 2012. ACM.

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

[10] J. Gao, W. Yuan, X. Li, K. Deng, and J.-Y. Nie. Smoothing clickthrough data for web search ranking. In *SIGIR*, pages 355–362. ACM, 2009.

[11] M. Girvan and M. E. Newman. Community structure in social and biological networks. *Proc Natl Acad Sci U S A*, 99(12):7821–7826, June 2002.

[12] B. H. Good, Y.-A. de Montjoye, and A. Clauset. The performance of modularity maximization in practical contexts. *PHYS.REV.E*, 81:046106, 2010.

[13] R. Kannan, S. Vempala, and A. Vetta. On clusterings: Good, bad and spectral. *J. ACM*, 51(3):497–515, 2004.

[14] A. Lancichinetti, S. Fortunato, and F. Radicchi. Benchmark graphs for testing community detection algorithms. *Phys. Rev. E*, 78(4):046110, Oct. 2008.

[15] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Statistical properties of community structure in large social and information networks. In *WWW*, pages 695–704, New York, NY, USA, 2008. ACM.

[16] J. Leskovec, K. J. Lang, and M. W. Mahoney. Empirical comparison of algorithms for network community detection. In *WWW*, 2010.

[17] H. Maserrat and J. Pei. Neighbor query friendly compression of social networks. In *KDD*, pages 533–542, New York, NY, USA, 2010. ACM.

[18] S. Navlakha, R. Rastogi, and N. Shrivastava. Graph summarization with bounded error. In *SIGMOD Conference*, pages 419–432, 2008.

[19] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review*, E 69(026113), 2004.

[20] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[21] C. H. Papadimitriou. The NP-completeness of the bandwidth minimization problem. *Computing*, 16(3):263–270, 1976.

[22] U. N. Raghavan, R. Albert, and S. Kumara. Near linear time algorithm to detect community structures in large-scale networks. *PHYSICAL REVIEW E*, 76:036106, 2007.

[23] P. Ronhovde and Z. Nussinov. Local resolution-limit-free potts model for community detection. *PHYSICAL REVIEW E*, 81:046114, 2010.

[24] M. Rosvall and C. T. Bergstrom. Maps of random walks on complex networks reveal community structure. *PNAS*, 105:1118, 2008.

[25] V. V. Vazirani. *Approximation algorithms*. Springer-Verlag New York, Inc., New York, NY, USA, 2001.