Neighbor Query Friendly Compression of Social Networks*

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

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

ABSTRACT

Compressing social networks can substantially facilitate mining and advanced analysis of large social networks. Preferably, social networks should be compressed in a way that they still can be queried efficiently without decompression. Arguably, neighbor queries, which search for all neighbors of a query vertex, are the most essential operations on social networks. Can we compress social networks effectively in a neighbor query friendly manner, that is, neighbor queries still can be answered in sublinear time using the compression? In this paper, we develop an effective social network compression approach achieved by a novel Eulerian data structure using multi-position linearizations of directed graphs. Our method comes with a nontrivial theoretical bound on the compression rate. To the best of our knowledge, our approach is the first that can answer both out-neighbor and in-neighbor queries in sublinear time. An extensive empirical study on more than a dozen benchmark real data sets verifies our design.

Categories and Subject Descriptors

E.2 [**Data**]: Data Storage Representations

General Terms

Algorithms, Experimentation, Theory

Keywords

Social Networks, Compression, MP_k linearization, Eulerian data structure

1. INTRODUCTION

A social network is a network of individuals or organizations as nodes, linked by one or multiple types of interdependency, such as friendship, financial relationships, and kinship. Recently, facilitated by World Wide Web, more and more online social networks have been formed on the Web, such as Facebook, Twitter, and LinkedIn. It has been well recognized that mining social network data can provide precious actionable knowledge for business and individuals. As an evidence, the number of US patent applications on new technologies related to social networks has grown exponentially since 2003 at a rate of about 250% per year (`http://en.wikipedia.org/wiki/File:Growth_in_Social_Network_Patent_Applications.jpg`).

Social networks are often large, and become even larger and larger. For example, from January 2008 to September 2008, the number of active users of Facebook grew from 60 millions to 140 millions (`http://venturebeat.com/2008/12/18/2008-growth-puts-facebook-in-better-position-to-make-money/`). Facebook has more than 350 million active users nowadays. The large size and the fast growth rate present a huge challenge for mining and analyzing large social networks.

Compressing social networks can substantially facilitate mining and advanced analysis of large social networks. Preferably, we want *query-able compression* of social networks. In other words, social networks should be compressed in a way that they still can be queried efficiently without decompression.

Many different kinds of queries can be conducted on social networks. Arguably, neighbor queries, which search for all neighbors of a query vertex, are the most essential operations on social networks. Many other operations, such as community finding, network pattern mining, and outlier detection, can be built based on neighbor queries.

Can we compress social networks effectively in a neighbor query friendly manner, that is, neighbor queries still can be answered in sublinear time using the compression? As to be analyzed in Section 2, this is a challenging task, and the existing Web or social network compression methods either have to maintain the compressions of both the network and its transpose, or cannot answer neighbor queries in sublinear time at all.

In this paper, we tackle the problem of neighbor query friendly compression of social networks. We develop an effective social network compression approach. Specifically, we propose a novel Eulerian data structure using multi-position linearizations of directed graphs. When the optimal MP_1 linearization is used, our method comes with a nontrivial

*This research is supported in part by an NSERC Discovery Grant and an NSERC Discovery Accelerator Supplement Grant. 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.

KDD'10, July 25–28, 2010, Washington, DC, USA.

theoretical upper bound on the number of bits used per edge in the compression. To the best of our knowledge, our approach is the first that can answer both out-neighbor and in-neighbor queries in sublinear time. Moreover, we explore effective extensions using MP_k linearization. An extensive empirical study on more than a dozen benchmark real data sets verifies our design.

The rest of the paper is organized as follows. In Section 2, we review the related work. In Section 3, we define the basic notions, formulate neighbor queries, and review the concepts of Eulerian paths and multi-position linearization. We present our novel Eulerian data structure in Section 4, and explore the extensions to using MP_k linearization in Section 5. We report an empirical study in Section 6, and conclude the paper in Section 7.

2. RELATED WORK

Due to the fast growth of WWW, compressing Web graphs has received substantial research interest. A Web graph typically contains a huge number of Web pages as vertices, and an even larger number of hyperlinks as directed edges.

Adler and Mitzenmacher [1] gave a Web graph compression method by finding nodes with similar sets of neighbors. Randall *et al.* [13] were the first to use the lexicographic ordering of URLs of Web pages for compressing the graph. Their method takes advantage of the fact that many hyperlinks are intra-host, and many pages on the same host have similar hyperlinks. Boldi and Vigna [4, 5] further exploited the properties of Web pages in lexicographic ordering to achieve better compression. Specifically, their method takes advantage of the lexicographic locality in Web graphs. That is, proximal pages in URL lexicographic order often have similar neighborhoods. For better compression, Boldi *et al.* [3] further developed new orderings combining host information and Gray/lexicographic orderings.

Orthogonal to the exploitation of lexicographic ordering, Raghavan and Garcia-Molina [12] decomposed a Web graph into a hierarchical structure. They used the notion of S-node to capture the locality property of Web graphs. Suel and Yuan [15] also used the structural decomposition technique in some sense by distinguishing between local and global links. Recently, Apostolico and Drovandi [2] introduced a BFS-based method. Their approach also encodes the gaps between links, but uses a more general setting.

Buehrer and Chellapilla [7] used a data mining approach to tackle the problem of compressing Web graphs. Using frequent item-set mining techniques they mined the complete bipartite subgraphs and replaced the edges of those subgraphs by a virtual node connecting to all vertices in both partitions in the complete subgraph. Their method, with the combination of gap coding technique in the lexicographic order, achieves the performance of under two bits per edge.

All the methods mentioned above use the bits/edge rate as the primary evaluation measure. Some of them also report the query processing performance. Specifically, Raghavan and Garcia-Molina [12] provided a direct comparison with the method by Randall *et al.* [13] on a collection of six different complex queries. Boldi and Vigna [4, 5] introduced the lazy iteration for (randomly and sequentially) accessing the links in a compressed web graph. The access time for their approach is in the order of several hundred nano seconds per link.

Most of the existing Web graph compression methods exploit the locality of links in the lexicographic order of web pages. Can social networks, which do not have a natural lexicographic order for the vertices, be compressible to the same degree? Recently, Chierichetti *et al.* [8] extended the Web graph compression framework [4, 5] by Boldi and Vigna (the BV schema) to compress social networks. The central idea is to introduce an ordering based on Jaccard coefficient [6]. By integrating this ordering into the BV schema, they introduced a compression schema for social networks. They further exploited the reciprocal edges. However, one drawback is that their method cannot answer neighbor queries in sublinear time.

Our study is also related to the family of graph layout (or graph ordering) problems, where the goal is to find an ordering of the nodes minimizing a given objective function. Díaz *et al.* [10] presented a nice survey. In particular, Papadimitriou [11] proved the NP-hardness of the minimum bandwidth problem, where the objective is to minimize the maximum stretch of the edges.

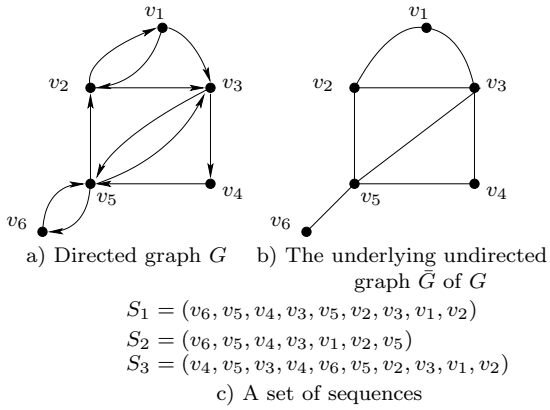


Figure 1: A directed graph and its underlying undirected graph

3. PRELIMINARIES

In this section, we review the essential notions used in the paper, define the neighbor queries of interest in this paper, discuss the background on Eulerian path, and introduce the notion of multi-position linearization for a graph.

3.1 Notions

In this paper, we model a social network as a *directed graph* $G = (V, E)$ where V is a set of vertices and $E \subseteq V \times V$ is a set of edges. We also refer to V by $V(G)$ and to E by $E(G)$. For an edge $e = (u, v)$, we refer to u as the *source* of e and v as the *destination* of e. $(u, v) \neq (v, u)$.

A *simple directed graph* is a directed graph such that there does not exist a self-loop, i.e., no edge (u, u) for any vertex u, and there is at most one edge from a source u to a destination v. In this paper, we consider simple directed graphs only. However, it is straightforward to generalize our results and algorithms to deal with directed graphs that contain self-loops and multiple edges between two vertices.

In an *undirected graph*, edges do not carry direction information, i.e., $\{u, v\} = \{v, u\}$. For a directed graph G, we can obtain the *underlying undirected graph* $\bar{G}$ of G such that $\{u, v\} \in E(\bar{G})$ if and only if $(u, v) \in E(G)$ or $(v, u) \in E(G)$. Figure 1 shows an example.

For an undirected graph G, we can obtain the *directed version* $G^{\rightleftarrows}$ of G by placing two directed edges (u, v) and (v, u) in $G^{\rightleftarrows}$ for each undirected edge $\{u, v\}$ in G.

Hereafter, we call a simple directed graph simply a graph if there is no ambiguity. Occasionally, we use the notion of undirected graphs which will be mentioned explicitly.

For a graph G, the *transpose* of G, denoted by G^T, is a graph such that $V(G^T) = V(G)$ and $(u, v) \in E(G^T)$ if and only if $(v, u) \in E(G)$.

In a graph G, an edge $(u, v) \in E$ is called *reciprocal* if $(v, u) \in E$ as well. In such a case, u and v are immediately connected in both directions. Let $Fre(G)$ be the fraction of reciprocal edges in $E(G)$, i.e.,

$$Fre(G) = \frac{\text{number of reciprocal edges in } E(G)}{|E(G)|}.$$

Therefore, $G = G^T$ if and only if $Fre(G) = 1$.

In an undirected graph $\bar{G}$, a vertex u is a *neighbor* of a vertex v if $\{u, v\} \in E(\bar{G})$. Let N_v be the set of neighbors of v and $E_v = \{\{u_1, u_2\} \in E(\bar{G}) | u_1, u_2 \in N_v\}$ be the set of edges between the vertices in N_v. For a directed graph G we use its underlying undirected graph $\bar{G}$ to define $Acc(G)$, the average clustering coefficient [16], as

$$Acc(G) = Acc(\bar{G}) = \frac{1}{|V(\bar{G})|} \sum_{v \in V(\bar{G})} \frac{2|E_v|}{|N_v|(|N_v| - 1)}$$

Moreover, we define $Gcc(G)$, the global clustering coefficient [14], as

$$Gcc(G) = Gcc(\bar{G}) = \frac{2 \sum_{v \in V(\bar{G})} |E_v|}{\sum_{v \in V(\bar{G})} |N_v|(|N_v| - 1)}$$

3.2 Neighbor Queries in Directed Graphs

In a directed graph G, there are two types of neighbors. For a vertex $u \in V(G)$, $v_1 \in V(G)$ is an *out-neighbor* of u if $(u, v_1) \in E(G)$. Moreover, $v_2 \in V(G)$ is an *in-neighbor* of u if $(v_2, u) \in E(G)$. An *out-neighbor query* on u is to find the set of out-neighbors of u. Similarly, an *in-neighbor query* on u is to search for all in-neighbors of u.

EXAMPLE 1 (NEIGHBOR QUERIES). *In Figure 1(a), an out-neighbor query on v_5 in G returns $\{v_2, v_3, v_6\}$. An in-neighbor query on v_5 returns $\{v_3, v_4, v_6\}$. Please note that v_3 and v_6 are both out-neighbors and in-neighbors of v_5, since there are reciprocal edges between v_3 and v_5 as well as between v_5 and v_6.*

All the existing methods for compressing Web graphs or social networks encode only outgoing edges. Consequently, most of those methods can only answer out-neighbor queries directly. In order to answer in-neighbor queries, they have to store a compressed version of the transpose of the graph. As mentioned in Section 2, some methods like [8] cannot answer neighbor queries in sublinear time.

3.3 Eulerian Paths

A *path* P of length k in a graph G is a sequence of edges $(u_1, u_2), (u_2, u_3), \ldots, (u_k, u_{k+1})$, where $(u_i, u_{i+1}) \in E(G)$ $(1 \leq i \leq k)$. For the sake of simplicity, we often write path P as $(u_1, u_2, \ldots, u_{k+1})$. P is a *simple path* if $u_1, \ldots, u_{k+1}$ are unique among one another.

DEFINITION 1 (EULERIAN PATH). *An **Eulerian path** for an undirected graph is a path in the graph which visits each edge of the graph exactly once.*

EXAMPLE 2 (EULERIAN PATH). *In Figure 1(b), path S_1 is an Eulerian path for $\bar{G}$, S_2 is not because it does not visit edge $\{v_5, v_3\}$, S_3 is not an Eulerian path, either, because it is not a path – $\{v_4, v_6\}$ is not an edge in graph $\bar{G}$.*

It is well known that a connected undirected graph G has an Eulerian path if and only if it has at most two vertices with odd degrees.

A simple algorithm to construct the Eulerian path which dates back to 1883, known as Fleury's algorithm, is as follows: we start with a vertex of an odd degree. If there is no such a vertex, we start with any vertex. At each step we move across an edge whose deletion does not disconnect the graph, unless there is no other choice. We repeat this process until no edge is left.

3.4 Multi-Position Linearization

In this subsection we introduce the notion of *multi-position linearization* of degree k (MP_k *linearization* for short) for a given graph G.

Let $S = (v_{i_1}, v_{i_2}, \ldots, v_{i_m})$ be a sequence of vertices of graph G (with possible replication). We say S *covers* G if all the vertices of G appear at least once in S. The length of S is m. Here, S does not need to be a path.

We need the following notion of S-distance.

DEFINITION 2 (S-DISTANCE). *Given a sequence S that covers a graph G, the **S-distance** between u and v, denoted by S-dist(u, v), is the minimum norm-1 distance among all pairs of appearances of u and v.*

EXAMPLE 3 (S-DISTANCE). *In Figure 1, the S_1-distance between v_3 and v_5 is S_1-dist$(v_3, v_5) = 1$. S_1-dist$(v_2, v_4) = 3$ and S_2-dist$(v_5, v_3) = 2$.*

Now we are ready to formally define the notion of MP_k linearization.

DEFINITION 3 (MP_k). *An **MP_k linearization** of a graph G is a sequence S of vertices of the graph with possible replication, such that S covers G and for all $(u, v) \in E(G)$, S-dist$(u, v) \leq k$. The **length** of an MP_k linearization is equal to length of S.*

If S is an MP_k linearization of a graph G, S is also an MP_k linearization of the underlying undirected graph $\bar{G}$, and vice versa.

EXAMPLE 4 (MP_k). *In Figure 1, the sequences S_1 and S_3 are two different MP_1 linearizations of both G and $\bar{G}$. But the length of S_1 is less than that of S_3. S_2 is an MP_2 linearization but not an MP_1 linearization of G, because for edge $(v_5, v_3) \in E$, S_2-dist$(v_5, v_3) = 2$.*

The notion of S-distance can be regarded as an embedding of the metric space defined by G to a simpler and computationally more efficient metric space.

4. EULERIAN DATA STRUCTURE

As mentioned in Section 3.2, in order to answer both out-neighbor queries and in-neighbor queries efficiently, most of the existing methods have to store both a graph G and its transpose G^T. Is there any schema for encoding directed graphs which can support fast out-neighbor and in-neighbor queries and still retain a good compression rate? In other

words, how can we encode the graph G and its transpose G^T at the same time using space less than keeping two independent copies of the edges?

In this section, we introduce the Eulerian data structure that affirmatively answers this question. Our method compresses a directed graph into an Eulerian data structure. Before that, we need to establish the connection between MP_1 linearization and Eulerian path of a graph.

4.1 MP_1 linearization and Eulerian path

The following proposition is immediate:

PROPOSITION 1 (LOWER BOUND OF MP_1 LINEARIZATION). *Given a directed graph G, the lower bound for the length of the MP_1 linearization of G is*

$$|E(\bar{G})| + 1 = (1 - \frac{Fre(G)}{2})|E(G)| + 1.$$

Moreover the bound is tight if and only if $\bar{G}$ has an Eulerian path, i.e. it has at most two vertices of odd degrees.

EXAMPLE 5 (LOWER BOUND OF MP_1 LINEARIZATION). *The lower bound for the length of MP_1 linearization of G in Figure 1 is 9, since $|E(\bar{G})| = 8$. We can write $|E(\bar{G})|$ in terms of $|E(G)|$ using $Fre(G)$. Since $Fre(G) = 6/11$, $|E(\bar{G})| = (1 - \frac{6/11}{2})|E(G)| = 8$.*

Unlike the Eulerian path construction problem which is an existence problem, finding the optimal MP_1 linearization is an optimization problem. No matter what structure a graph has, always there exists an optimal (i.e., shortest) MP_1 linearization for the graph. The following lemma gives the length of an optimal MP_1 linearization of an arbitrary directed graph.

LEMMA 1. *The minimum length of MP_1 linearization of an arbitrary directed graph G is $|E(\bar{G})| + \max\{n_{odd}/2, 1\}$, where n_{odd} is the number of vertices with odd degrees in $\bar{G}$.*

PROOF. In any undirected graph $\bar{G}$, the sum of degrees of all vertices is even. Thus, the number of vertices with odd degrees is also even.

We first prove by induction that for any graph G we can obtain an MP_1 linearization of length $|E(\bar{G})| + \max\{n_{odd}/2, 1\}$.

Basis: For $n_{odd} = 0$ and $n_{odd} = 2$, the claim follows from proposition 1.

Induction: Since n_{odd} must be even, we assume that the lemma holds for $n_{odd} = 2k$ ($k \geq 1$), and consider the case when $n_{odd} = 2(k+1)$. There are two subcases.

In the first subcase, there are two vertices u and v with odd degrees such that they are not connected in $\bar{G}$. We add an edge $\{u, v\}$ to $\bar{G}$ and call the new graph $\bar{G}_*$. Since $\bar{G}_*$ has only $2k$ vertices with odd degrees, applying the induction assumption, we can obtain an MP_1 linearization of $\bar{G}_*$ with length $|E(\bar{G}_*)| + k = |E(\bar{G})| + k + 1$. Since $E(\bar{G}) \subset E(\bar{G}_*)$ the MP_1 linearization of $\bar{G}_*$ is indeed an MP_1 linearization for $\bar{G}$.

In the second subcase, all vertices with odd degrees are connected. We arbitrarily take two vertices u and v with odd degrees and remove the edge $\{u, v\}$ from $\bar{G}$. Let us call the resulting graph $\bar{G}_*$. Again, $\bar{G}_*$ has $2k$ vertices with odd degrees. Therefore, there is an MP_1 linearization with length $|E(\bar{G}_*)| + k = |E(\bar{G})| - 1 + k$ for $\bar{G}_*$. Since the MP_1 linearization for $\bar{G}_*$ does not cover $\{u, v\}$, we have to add u and v to the end of the linearization. Therefore we just build an MP_1 linearization for $\bar{G}$ of length $|E(\bar{G})| + k + 1$, as desired.

Now we prove that $|E(\bar{G})| + \max\{n_{odd}/2, 1\}$ is also a lower bound for the MP_1 linearizations for G. For $n_{odd} = 0$ and $n_{odd} = 2$, simply applying Proposition 1 gives the bound.

Now, let us consider the case where $n_{odd} \geq 4$. Let $v(i)$ be the vertex that appears in the position i of an MP_1 linearization L. For a vertex u with an odd degree in $\bar{G}$ which appears in neither the first nor the last position of L, we denote by $P_u = \{i | v(i) = u\}$ the set of all appearances of u in L. There are in total at least $n_{odd} - 2$ such vertices.

For any interior position i in L, there are two edge slots in L: $(i-1, i)$ and $(i, i+1)$. Consider all the edge slots associated with the positions in P_u. At least one of these slots must be a "waste", that is, there is no edge appearing in the slot or the edge in the slot also appears in some other slot. Otherwise, the degree of u is $2|P_u|$, which is an even number.

In the best scenario, two vertices with odd degrees can share a wasted edge slot. Therefore, we have at least $(n_{odd} - 2)/2$ wasted edge slots. In addition, we need $|E(\bar{G})|$ edge slots to cover the edges of $\bar{G}$. Therefore, in total L has to have at least $|E(\bar{G})| + n_{odd}/2 - 1$ edge slots. Hence, the length of L cannot be smaller than $|E(\bar{G})| + \max\{n_{odd}/2, 1\}$.

Please note that the induction in the proof of Lemma 1 also gives an algorithm to find an optimal MP_1 linearization of a graph G. Since the complexity of finding an Eulerian path is $O(|E(G)|)$ [9], finding an optimal MP_1 linearization of a graph G is also of the same complexity.

EXAMPLE 6. *In Figure 2, $\bar{G}_2$ has 6 vertices of odd degrees, namely v_1, v_2, v_4, v_5, v_7 and v_8. Therefore, the lower bound on the length of MP_1 linearization is $15 + 6/2 = 18$. Indeed the MP_1 linearization of Figure 2 is optimal.*

4.2 The Eulerian Data structure

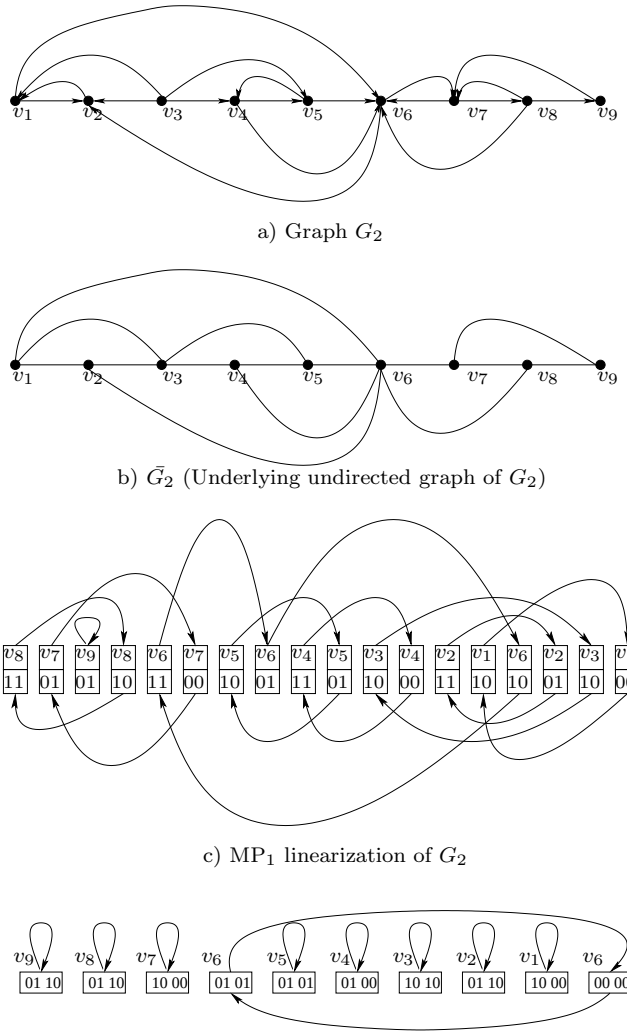
Based on the relation between Eulerian path and MP_1 linearization, we present a novel data structure for encoding graphs. To keep our discussion simple, similar to [4, 5, 8], we assume (1) we are allowed to renumber the vertices; (2) for each vertex there is an identifier which can be used for referring to the vertex. However, our data structure does not maintain an index of the identifiers; and (3) the edges are not labeled. Please note that we can straightforwardly extend the data structure to remove the above assumptions.

DEFINITION 4 (EULERIAN DATA STRUCTURE). *The **Eulerian data structure** for a graph G stores an optimal MP_1 linearization L of G using an array of the same length as L. Let $v(i)$ be the vertex in G that appears at the position i of L. For cell i of the Eulerian data structure, we keep the following two pieces of information*

***Local information:** two bits specifying if edges $(v(i-1), v(i))$ and $(v(i), v(i-1))$ belong to $E(G)$, respectively.*

***Pointer:** a pointer to the next appearance of $v(i)$. If this is the last appearance of $v(i)$, then the pointer points to the first appearance of the vertex.*

EXAMPLE 7 (EULERIAN DATA STRUCTURE). *In Figure 2(c), the Eulerian data structure of G in Figure 2(a) using an optimal MP_1 linearization is illustrated. Here we show the pointers by arcs. Since the length of the linearization is 18, we need $\lceil \log_2 18 \rceil = 5$ bits to encode*



a) Graph G_2

b) $\bar{G}_2$ (Underlying undirected graph of G_2)

c) MP_1 linearization of G_2

d) MP_2 linearization of G_2

Figure 2: MP linearizations of an arbitrary directed graph.

each pointer. Therefore, for each position we need $5+2$ *bits. In total we need* $18\times(5+2)=126$ *bits, which make a compression rate of* $128/19\approx 6.63$ *bits per edge.*

We have the following important result on the compression efficiency of the Eulerian data structure.

THEOREM 1. *An Eulerian data structure to encode a graph G uses up to*

$$(1-\frac{Fre(G)}{2}+\frac{1}{\bar{d}})\Big(\lceil\log_2(|V(\bar{G})|)+\log_2(\bar{d}+1)\rceil+1\Big)$$

bits per edge on average, where $\bar{d}$ is the average degree of $\bar{G}$. Moreover, using this data structure, it is possible to answer the in-neighbor and out-neighbor queries for any vertex v in $O(\sum_{u\in N_v} deg(u)\log|V(G)|)$ time, where $deg(u)$ is the degree of vertex u in $\bar{G}$ and N_v is the set of out-neighbors/in-neighbors (resp.) of v in out-neighbor/in-neighbor queries.

PROOF. Let L be an optimal MP_1 linearization of $\bar{G}$ (therefore for G as well). Since there are at most $|V|$ vertices of odd degrees in $\bar{G}$, the upper bound for the length of L is $|E(\bar{G})|+|V(\bar{G})|/2$. Using 2 bits to store the local information and $\lceil\log_2(|E(\bar{G})|+|V(\bar{G})|/2)\rceil$ bits for the pointer for each cell, in total the Eulerian data structure uses at most

$$\Big(|E(\bar{G})|+|V(\bar{G})|/2\Big)\Big(2+\lceil\log_2(|E(\bar{G})|+|V(\bar{G})|/2)\rceil\Big)$$

bits. To get the bits/edge rate we divide this by the number of edges of G, and have

$$\Big(\frac{|E(\bar{G})|+|V(\bar{G})|/2}{|E(G)|}\Big)\Big(2+\lceil\log_2(|E(\bar{G})|+|V(\bar{G})|/2)\rceil\Big).$$

Notice that we can write the ratio of $|E(\bar{G})|/|E(G)|$ in terms of $Fre(G)$ and also

$$\frac{|V(\bar{G})|}{2|E(G)|}\leq\frac{|V(\bar{G})|}{2|E(\bar{G})|}$$

which is precisely the inverse of the average degree of $\bar{G}$. Therefore, the bits/edge rate is at most:

$$\Big(1-\frac{Fre(G)}{2}+\frac{1}{\bar{d}}\Big)\Big(2+\lceil\log_2(|E(\bar{G})|+|V(\bar{G})|/2)\rceil\Big)$$

We use $|E(\bar{G})|=\frac{|V(\bar{G})|\bar{d}}{2}$ to further simplify the inside of the logarithm, and obtain

$$\begin{aligned}&\Big(1-\tfrac{Fre(G)}{2}+\tfrac{1}{d}\Big)\Big(2+\lceil\log_2(|V(\bar{G})|\bar{d}/2+|V(\bar{G})|/2)\rceil\Big)\\&=\Big(1-\tfrac{Fre(G)}{2}+\tfrac{1}{d}\Big)\Big(2+\lceil\log_2(|V(\bar{G})|)+\log_2(\bar{d}+1)-\log_2(2)\rceil\Big)\end{aligned}$$

The upper bound is proved.

Since each vertex u of G appears in at least one position in the Eulerian data structure, we use the first position of u in L as the identifier for the vertex. Therefore, for an out-neighbor/in-neighbor query on vertex u, we have to return the positions of the first appearances of all out-neighbors/in-neighbors of u. Fetching the local information (only two bits) for each position takes constant time. Reading the pointer takes $O(\log|V(G)|)$ time (the number of bits for each pointer). Since the length of the linked list of positions for a vertex u is $\lceil\log(deg(u))\rceil$ in the MP_1 linearization of G, traversing over the linked list takes $O(deg(u)\log_2|V(G)|)$ time. By traversing the linked list of u we can retrieve the positions of all neighbors of u. However, for a neighbor v of u, the retrieved position may not be the first appearance of v. Therefore, for each retrieved neighbor v, we have to traverse the linked list for v to get the first appearance. So, answering an out-neighbor/in-neighbor query takes $O(\log(|V(G)|)\sum_{u\in N_v} deg(u))$ time in total.

As a baseline for representing a graph G with sublinear in-neighbor and out-neighbor queries, we can use $2\lceil\log_2|V(G)|\rceil$ bits to encode an edge. For social networks in practice, it is reasonable to assume that the average degree increases logarithmatically with respect to the number of nodes in the graph. Therefore, asymptotically (i.e., assuming the number of nodes approaches infinity) the Eulerian data structure uses half of the number of bits that the baseline schema uses due to the following equation.

$$\begin{aligned}&\lim_{|V(G)|\to\infty}\frac{(1+\frac{1}{\log_2|V(G)|})(\log_2(|V(G)|)+\log_2\log_2(|V(G)|)+1)}{2\log_2(|V(G)|)}\\&=\frac{1}{2}\end{aligned}$$

Average degree	Number of vertices			
	10^3	10^5	10^7	10^9
10	0.83	0.71	0.66	0.64
100	0.91	0.74	0.67	0.64
500	1.00	0.80	0.71	0.67

Table 1: Comparison of compression using the Eulerian data structure against the baseline schema.

Table 1 compares our method and the baseline schema for a number of combinations of $\bar{d}$ and $|V(G)|$. Clearly, the larger and the sparser the graph, the less bits the Eulerian data structure uses. Real life social networks are often large and sparse. Thus, the Eulerian data structure is capable of compressing social networks.

To the best of our knowledge, the Eulerian data structure is the first schema that allows answering both out-neighbor and in-neighbor queries in sublinear time, and provides a nontrivial theoretical upper bound on the number of bits per edge. The upper bound given by Theorem 1 is for arbitrary graphs, including totally random graphs. We know that for some subclasses of graphs the information theoretic lower bound is $\lceil \log N \rceil$ bits per edge. Therefore, from a theoretical point of view our upper bound is not very far away from this information theoretic lower bound, at least asymptotically close.

The real world networks in many cases exhibit some kind of locality property which can be used to further improve our method. We will detail our methods in the next section.

5. COMPRESSION USING MP_K LINEARIZATION

A natural extension for the Eulerian data structure is to use MP_k linearization instead of MP_1 linearization. This raises several new challenges. First of all, unlike MP_1 linearization, finding an optimal MP_k linearization for $k \geq 2$ is NP-hard in general, since it can be regarded as a generalization of the minimum bandwidth problem [11]. How can we get a "good" MP_k linearization without much cost? Second, given an MP_k linearization, we need to store $2k$ bits as the local information for each position i to record whether (i, j) and (j, i) are edges in the graph for $|i - j| \leq k$. The amount of local information is considerable. Hence, storing the local information efficiently is important. To address the above issues, we extend our method to tackle two subproblems: MP_k linearizing a graph and encoding the local information for each position of the MP_k linearization.

In this section, we first motivate the extension. Then, we discuss the tradeoff. Finally, we present the heuristics and algorithms.

5.1 Motivation of Using MP_k

The most commonly used measure for locality in social networks is average clustering coefficient. In social networks, the number of vertices with a small degree is much more than the number vertices with a large degree. Consequently, average clustering coefficient usually has a bias toward the vertices with small degrees. Therefore, we consider the measure *Gcc*, Global clustering coefficient, as well.

Roughly speaking, *Gcc* measures the probability that there is an edge between two vertices when they have a common neighbor. Consider a social network with a large *Gcc* value. Suppose vertices u, v, w and t are four consecutive vertices in an MP_k linearization, and there are edges between u and v, v and w, as well as w and t. There is a good chance that u and w are connected, since both have v as a neighbor. Moreover, if u and w are connected, using the same argument again, there is a good chance that u and t are also connected. Depending on how strong the locality of the network is, it does make sense to keep more bits for u specifying weather $\{u, w\}, \{w, u\}, \{u, t\}$ and $\{t, u\}$ belong to $E(\bar{G})$ or not.

5.2 Tradeoffs

There is a tradeoff between the length of linearization against the amount of local information. The tradeoff highly depends on the structure of the graph. Intuitively, for a large sparse graph where each vertex has the same out degree, and the destinations of the out-edges are picked randomly, increasing k would not influence the length of the linearization significantly. However, for a large random dense graph G where the existence of an edge from every node to another is independently determined by a probability of 50%, increasing k up to $|V(G)|$, the number of vertices, is actually beneficial.

EXAMPLE 8. *Figures 2(c) and (d) encode graph G_2 in Figure 2(a) using an MP_1 linearization and an MP_2 linearization, respectively. Using the MP_1 linearization, we need to use 2 bits to store the local information for each position. Since the MP_1 linearization has 18 positions, we need to use 5 bits for each pointer. In total we need $18 \times (5 + 2) = 126$ bits, approximately 6.63 bits per edge on average.*

Using the MP_2 linearization, we need to use 4 bits to store the local information for each position. Since it has 10 positions, we need to use 4 bits for each pointer. In total it uses $10 \times (4 + 4) = 80$ bits, approximately 4.21 bits per edge. The saving of using the MP_2 linearization is substantial.

Can we save more by moving from MP_2 to MP_3 linearization? The length of an MP_k linearization for any k cannot be less that the number of vertices in the graph. G_2 has 9 edges. Therefore, in the best case of using an MP_3 linearization, we have 9 positions. For each position we have to use 6 bits to store the local information and 4 bits for the pointer. Thus, using an MP_3 linearization needs to use at least $9 \times (6 + 4) = 90$ bits. Using MP_3 linearization cannot save, comparing to using the MP_2 linearization.

5.3 Heuristics and algorithms

To ensure that we can handle large social networks, we use a straightforward greedy heuristic for linearizing a graph. The algorithm is shown in Algorithm 1. We start with a random vertex. At each step we append to the list the vertex that has the largest number of edges with the last k nodes in the list. We remove these edges from the graph and iterate until no edge is left. If none of the last k vertices in the list have a neighbor, then we pick a random node with non-zero degree and continue from there.

Using $2k$ bits we can encode the local information for each position. A practical problem of the greedy linearization heuristic is that, as we are removing the edges of the graph, the graph becomes sparser and sparser. Thus, having a fixed k all the time is not a good idea since the rear part of the linearization may have very few new edges to encode. To be adaptive, we use a relaxed version of the linearization notion. We start with a relatively large value of k (say 20) and watch

Name	Description	$\|V\|$	$\|E\|$	Acc	Gcc	Fre
amazon0302	Amazon product co-purchasing network from march 2, 2003	262111	1234877	0.424	0.236	0.542
amazon0312	Amazon product co-purchasing network from march 12, 2003	400727	3200440	0.411	0.160	0.531
ca-CondMat	collaboration network of Arxiv Condensed Matter	23133	186878	0.633	0.264	1
ca-HepPh	Collaboration network of Arxiv High Energy Physics	12006	236978	0.611	0.659	1
cit-HepPh	Arxiv High Energy Physics paper citation network	34546	421534	0.296	0.145	0.003
cit-Patents	Citation network among US Patents	3774768	16518947	0.091	0.067	0
email-Enron	Email communication network from Enron	36692	367662	0.497	0.085	1
email-EuAll	Email network from a EU research institution	265009	418956	0.309	0.004	0.260
p2p-Gnutella08	Gnutella peer to peer network from August 8 2002	6301	20777	0.015	0.020	0
p2p-Gnutella24	Gnutella peer to peer network from August 24 2002	26518	65369	0.009	0.004	0
soc-Slashdot0902	Slashdot social network from February 2009	82168	870161	0.061	0.024	0.841
soc-LiveJournal1	LiveJournal online social network	4846609	68475391	0.312	0.288	0.374
web-Google	Web grpah from Google	875713	5105039	0.604	0.055	0.306
web-Stanford	Web graph of Stanford.edu	281903	2312497	0.610	0.096	0.276

Table 2: The dataset stats. (Acc, Gcc, and Fre are defined in Section 3.1)

Algorithm 1 Find an MP_K linearization of G

Input: K, reducing factor RF ($0 \leq RF \leq 1$), density threshold DT ($0 \leq DT \leq 1$) and Graph G
Output: Linearization L of G
Description:
1: initialize L to an empty list
2: **while** $|E(G)| \geq 1$ **do**
3: let u be a random node with nonzero degree
4: append u to L
5: /* let X be the set of the last K vertices in L */
6: **while** X has at least one neighbor in $V(G) - X$ **do**
7: let v be the node which has the most number of edges to and from X
8: remove all edges between v and vertices in X
9: $edgecount+ = deg_{old}(v) - deg_{new}(v)$
10: append v to L
11: **if** Length(L)%1000==0 **then**
12: **if** $edgecount/2 * K * 1000 < DT$ **then**
13: $K = K * RF$
14: **end if**
15: $edgecount = 0$
16: **end if**
17: **end while**
18: **end while**

the average local density for the recent positions in the list (the last 1000 positions as shown in Algorithm 1). Once it drops below a certain density threshold DT, we reduce k by multiplying it to a predefined reducing factor RF.

We choose to use a simple heuristic for linearization and encode the local information. Our purpose is to examine the feasibility of the framework of using MP_k linearization for compressing social networks. Our method leaves space for further improvement which is the subject for future work.

6. EXPERIMENTS

To the best of our knowledge, there is no any existing social network compression method which can answer out-neighbor and in-neighbor queries in sublinear time. However, the existing methods which can answer out-neighbor queries can be made comparable to ours in functionality by encoding a given graph G and also its transpose G^T.

6.1 Experimental Setup

We used the data sets from the SNAP project (Stanford Network Analysis Package, `http://snap.stanford.edu/data/`). The data sets in the SNAP project are organized in different categories. From each category we chose the data sets with the smallest and the largest *Gcc* values, respectively, in order to test the effect of our method with respect to social networks of different degrees of locality. Those data sets are from very different domains, such as social networks, web graphs, peer-to-peer networks, collaborative networks, citation networks, and co-purchasing networks. Table 2 provides the statistics of these networks and short descriptions.

We implemented our algorithms using C++, on top of the SNAP library which is publicly available at `http://snap.stanford.edu/`. We used a heterogeneous linux based cluster to run most of the experiments. To report the running time, we selected a subsets of our experiments and ran them on a core(TM)2 Duo 2.66GHz linux system with 2GB of main memory.

Our method has three parameters: Reducing Factor (RF), (Starting) neighborhood size (K) and Density Threshold (DT). The last two parameters are more important than the first one, since they have direct control on the linearization generated. Therefore, we conducted an extensive experimental study on different values of these two parameters for each network in our collection. Particularly we are interested in the tradeoff between the length of the linearization and the neighborhood size.

We measured the compression performance using the bits/edge rate, as the previous studies did. In addition, we also report some other performance statistics such as query processing time.

Another interesting tradeoff in our method is between the out-neighbor query processing time and in-neighbor query processing time. An implementation decision is how to store the local information for each position. There are two options. In the first option, for each position in the Eulerian data structure, we can use the first k bits to record the out-edges to the previous k vertices in the linearization list, and use the next k bits to record the out-edges to the next k vertices in the list. In the second option, we can use the $2k$ bits to record both the out-edges and in-edges between the current position and the next k positions.

The first option biases on the out-neighbor queries. To answer an in-neighbor query about a vertex u, we have to scan

(K, reducing factor)	(10, 1)	(10, 0.9)			(15, 0.9)			(20, 0.9)			(30, 0.9)		
Density threshold	0	0.15	0.25	0.30	0.15	0.25	0.30	0.15	0.25	0.30	0.15	0.25	0.30
amazon0302	15.38	14.61	13.99	14.43	15.08	13.97	14.16	15.09	13.98	14.49	15.39	14.07	14.49
amazon0312	14.35	13.32	12.70	12.79	13.57	12.74	12.84	13.92	12.73	12.90	14.08	12.79	12.86
ca-CondMat	7.89	7.69	6.96	6.69	8.35	7.16	6.77	8.94	7.33	6.93	9.55	7.56	7.26
ca-HepPh	5.24	5.09	4.76	4.63	5.00	4.59	4.57	5.20	4.65	4.53	5.51	4.79	4.69
cit-HepPh	17.07	15.65	14.59	14.23	15.99	14.69	14.29	16.47	14.85	14.31	16.97	15.02	14.48
cit-Patents	31.59	27.69	25.95	25.75	27.63	25.97	25.69	27.73	25.95	25.69	27.78	25.97	25.78
email-Enron	8.72	8.11	7.39	7.26	8.53	7.47	7.27	8.88	7.52	7.31	9.19	7.64	7.44
email-EuAll	30.73	25.31	22.96	22.55	25.63	22.97	22.55	25.56	22.97	22.61	25.81	23.11	22.72
p2p-Gnutella08	30.36	25.48	22.90	21.63	26.70	23.88	23.42	29.82	27.13	26.88	33.84	33.21	33.21
p2p-Gnutella24	35.76	29.51	25.59	24.33	28.67	25.69	24.93	29.41	26.90	26.02	31.25	28.94	28.10
soc-Slashdot0902	16.17	14.19	12.68	12.14	14.55	12.69	12.15	14.63	12.68	12.17	14.75	12.74	12.19
soc-LiveJournal1	16.13	14.48	13.96	13.97	14.50	13.92	13.93	14.49	13.95	13.93	14.56	13.91	13.95
web-Google	12.84	12.22	11.63	11.66	12.29	11.58	11.68	12.74	11.61	11.70	12.99	11.59	11.65
web-Stanford	10.79	10.27	10.17	10.76	10.19	10.23	10.41	10.14	10.05	10.22	10.19	9.88	9.92

Table 3: The average number of bits per edge. The worse cases happen on those data sets that have very poor locality measures (Gcc and Fre)

the k positions preceding and following every occurrence of u in the linearization list. We implemented the second option in our experiments which does not bias on any specific types of neighbor queries.

6.2 Compression Rates

Table 3 summarizes the results about compression rate. While the performance of our method varies on different data sets, the interesting observation here is the strong negative correlation between the bits/edge rate and the value of locality measures. The average degree of the network seems important, too. In particular, Fre and Gcc are larger in Amazon0302 than in Amazon0312, but the performance of our method is better on Amazon0312. We believe that this is due to the higher average degree in Amazon0312 than Amazon0302.

It is interesting to look at the difference between data sets email-Enron and email-EuAll from the same category. Data set email-Enron has one of the best bits/edge rates and email-EuAll has one of the worst. We think this may be a footprint of the difference in communication patterns in industry and in academia.

The results clearly shows that our method takes advantage of the locality properties of the social networks. Our best result for the LiveJournal data set is 13.91 bits/edge, while the best result of BV scheme for the same data set is 14.308 (reported in [8]). Please note that BV scheme supports only the out-neighborhood queries. To answer both out-neighbor and in-neighbor queries, BV schema needs 2×14.308 bits per edge, assuming that encoding the transpose of the graph has approximately the same rate. Moreover, our method is flexible for incremental updates. We only need to encode the incremental subgraph. BV schema does not allow sublinear updates.

6.3 Query Processing Time

We report the query processing time for two types of queries. An adjacency query checks whether a query edge $(u, v) \in E$. A neighbor query searches for all out-neighbors and in-neighbors of a query vertex u.

We used $K = 20, RF = 0.25$ and $DT = 0.9$ as the default values for the parameters. Table 4 reports the average access time for the adjacency queries performed on the compressed graphs (comp.) and on the original graphs (SNAP) using the SNAP implementation of the graph data structure. We ran 1 million adjacency queries and 1 million neighborhood queries, and normalized the time by the number of edges that those queries returned. The time is in nano second.

	adj queries(ns)		Neigh. queries(ns)	
dataset	comp.	SNAP	comp.	SNAP
amazon0302	800	750	951	72
amazon0312	1170	790	1753	46
ca-CondMat	390	420	777	30
ca-HepPh	520	400	1849	19
cit-HepPh	1300	480	2745	28
cit-Patents	1400	930	1842	91
email-Enron	620	500	5539	31
email-EuAll	530	670	21518	148
p2p-Gnutella08	640	320	1663	34
p2p-Gnutella24	600	320	1488	50
soc-LiveJournal1	3050	1130	9734	49
soc-Slashdot0902	1380	610	7884	35
web-Google	810	830	4110	66
web-Standford	890	810	39939	49

Table 4: The average access time per edge for processing adjacency queries and (in+out) neighbor queries.

Our method spends up to 3 times more time to answer an adjacency query than that on the original graph. In most cases, extra cost in our method is very minor. For neighbor queries, the query answering time depends on the efficiency of the linearization. One vertex and one edge may appear multiple times in a linearization. The more replicates, the longer the query answering time.

6.4 Tradeoff between Local Information and Pointers

We divide the bits/edge rate in our method into two parts the bits/edge rate encoding local information, and that encoding the points. The total bits/edge rate is simply the sum of the two.

We studied the tradeoff between the local bits/edge rate

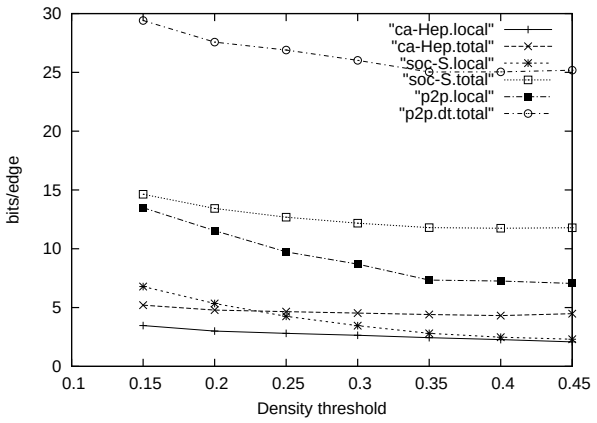


Figure 3: The trade off between the bits/edge rate of local information and that of pointers on three data sets: ca-HepPh, p2p-Gnutella24, and soc-Slashdot0902. ($K = 20$, $RF = 0.9$)

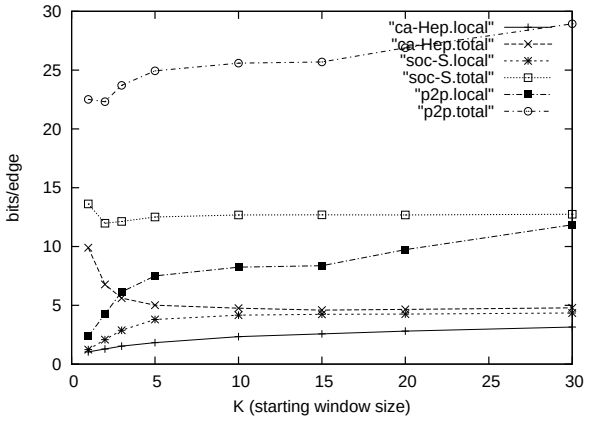


Figure 4: The trade off between the bits/edge rate of local information and that of pointers on three data sets: ca-HepPh, p2p-Gnutella24, and soc-Slashdot0902. ($DT = 0.25$ and $RF = 0.9$)

and that of the points when we varied the parameters of our method. Limited by space, we only report here the tradeoff for two parameters: the density threshold (DT) and the starting window size (K). We chose three data sets: ca-HepPh which has the best compression rate, p2p-Gnutella24 which has the worse compression rate, and soc-Slashdot0902 which has about the average compression rate.

In the first experiment, we varied $DT = 0.15$ to 0.45 with step 0.05, and fixed the other two parameters $K = 20$ and $RF = 0.9$. Figure 3 shows the local information bits/edge rate and the total bits/edge rate. Clearly, the compression rate is insensitive to parameter DT. Therefore, setting the parameter is not a big problem.

In the second experiment, we fixed $DT = 0.25$ and $RF = 0.9$, and varied K from 1 to 30. Figure 4 shows the results. Increasing k leads to better compression rates on the ca-HepPh and Soc-Slashdot0902 data sets. However, when k is 5 or larger, increasing k does not gain big advantage. Therefore, setting k to a value between 5 and 10 is a good experience choice.

7. CONCLUSIONS

In this paper, we tackled the problem of compressing social networks in a neighbor query friendly way. We developed an effective social network compression approach achieved by a novel Eulerian data structure using multi-position linearizations of directed graphs. Importantly, our method comes with a nontrivial theoretical bound on the compression rate. To the best of our knowledge, our approach is the first that can answer both out-neighbor and in-neighbor queries in sublinear time. An extensive empirical study on more than a dozen benchmark real data sets justifies the effectiveness of our method.

The encouraging results in this study suggest several interesting future directions. First, it is interesting to explore approximation methods for MP_k linearization for $k > 1$. Second, it is interesting to explore effective methods to determine a good value of k for MP_k linearization compression of social networks. Last, our heuristic algorithm is simple. It leaves space for further improvement in both the compression rate and the compression runtime.

8. REFERENCES

[1] M. Adler and M. Mitzenmacher. Towards compressing web graphs. In *Data Compression Conference*, 2001.

[2] A. Apostolico and G. Drovandi. Graph compression by BFS. *Algorithms*, 2(3):1031–1044, 2009.

[3] P. Boldi, M. Santini, and S. Vigna. Permuting web graphs. In *Proceedings of the 6th International Workshop on Algorithms and Models for the Web-Graph (WAW'09)*, Berlin, Heidelberg, 2009. Springer-Verlag.

[4] P. Boldi and S. Vigna. The webgraph framework I: compression techniques. In *WWW*, 2004.

[5] P. Boldi and S. Vigna. The webgraph framework II: Codes for the world-wide web. In *Data Compression Conference*, 2004.

[6] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *J. Comput. Syst. Sci.*, 60(3):630–659, 2000.

[7] G. Buehrer and K. Chellapilla. A scalable pattern mining approach to web graph compression with communities. In *WSDM*, 2008.

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

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001.

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

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

[12] S. Raghavan and H. Garcia-Molina. Representing web graphs. In *ICDE*, 2003.

[13] K. H. Randall, R. Stata, J. L. Wiener, and R. Wickremesinghe. The link database: Fast access to graphs of the web. In *DCC*, 2002.

[14] K. F. Stanley Wasserman. *Social networks analysis: Methods and Applications*. Cambridge Press, 1994.

[15] T. Suel and J. Yuan. Compressing the graph structure of the web. In *Data Compression Conference*, 2001.

[16] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small world' networks. *Nature*, 393(6684):440–442, 1998.