Social Network Compression: Not Only Space Saving, but also Insight Gaining

Jian Pei

Simon Fraser University
Email: jpei@cs.sfu.ca
http://www.cs.sfu.ca/~jpei

Joint work with Hossein Maserrat (my graduated Ph.D. student)

Outline

Social Networks in Our Life

- *"I have more high schools friends on Facebook than I ever had in high school?!"*
- *"God saw Adam was bored and lonely and sent Eve. God saw men and women were bored and sent Twitter"*

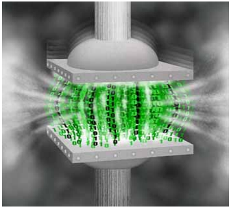
http://www.toonpool.com/cartoons/Social%20network_53133#

Social Networks Can Be Huge

- "*This morning, there are more than one billion people using Facebook actively each month, . . . Helping a billion people connect is amazing, humbling and by far the thing I am most proud of in my life.*"
  — Mark Zuckerberg on October 4, 2012
  - 1.11 billion users on May 1, 2013
- As of August 21, 2013, LinkedIn has more than 238 million registered members in over 200 countries and territories. (`http://press.linkedin.com/about`)
- As of September 2012, Twitter has 517 million registered users, 262 million active users, and even 35.5 million users in China. Twitter "still has more users there than any other country in the world, including the United States."
  — "Defying wisdom, report says Twitter is biggest in China" by Daniel Terdiman, October 5, 2012

Compressing Social Networks

- Analyzing huge social networks is great, only if we can handle them
  - Storage cost
  - Query answering cost
- Compressibility of a social network is a feature reflecting the structural characteristics of the social network
  - Compressibility of the whole social network
  - Compressibility of regions in a social network



Outline

Graph and Network Compression: Two Major approaches

- Aggregation based methods: using a “super-node” to replace a set of nodes that have similar neighbors
  - S. Navlakha, *et al.* Graph summarization with bounded error. In *SIGMOD'08*.
  - S. Raghavan and H. Garcia-Molina. Representing web graphs. In *ICDE'03*.
  - G. Buehrer and K. Chellapilla. A scalable pattern mining approach to web graph compression with communities. In *WSDM'08*.

Graph and Network Compression: Two Major approaches

- Aggregation based methods: using a “super-node” to replace a set of nodes that have similar neighbors
  - S. Navlakha, *et al.* Graph summarization with bounded error. In *SIGMOD'08*.
  - S. Raghavan and H. Garcia-Molina. Representing web graphs. In *ICDE'03*.
  - G. Buehrer and K. Chellapilla. A scalable pattern mining approach to web graph compression with communities. In *WSDM'08*.
- Ordering based methods: ordering nodes so that similar nodes fall into proximate positions
  - P. Boldi and S. Vigna. The webgraph framework I: compression techniques. In *WWW'04*.
  - F. Chierichetti, *et al.* On compressing social networks. In *KDD'09*.
  - P. Boldi, *et al.* Layered label propagation: a multiresolution coordinate-free ordering for compressing social networks. In *WWW'11*.

How Are Social Networks Different from Web Graphs?

- No natural ordering of vertices for general social networks
- Chierichetti *et al.* [KDD'09] used shingle ordering to compress social networks
- Shingle ordering tends to place nodes with similar out-links list close to each other (similar in the sense of Jaccard Coefficient)
- The compression rate in social networks tends to be not as good as that in Web graphs

Query Preserving Graph Compression

- Given a class of queries, compute the equivalence classes of nodes accordingly
- Build a smaller graph that has the equivalence classes as the vertices, which can be used to answer queries with quality guarantee
- Effective for simple queries, such as reachability, but less effective for more complex queries, such as pattern matching
- Not preserving community
- W. Fan *et al.* Query preserving graph compression. In SIGMOD'12.

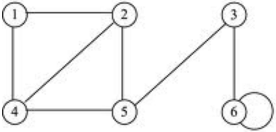
Outline

Goals

- Compressing social networks both globally and locally
  - Global compression: given a space budget, retaining as much information as possible
  - Local compression: communities are compressed in proximation so that they can be accessed locally in compression — using the compressed data without decompressing
- Compressibility as a structural property measure
  - Natural for community detection and quality assessment
  - Promising for visualization, summarization, and interactive analytics

When Are Adjacency Matrices Good?

- Adjacency matrices are often used to represent graphs
    - The adjacency matrix representation is often regarded inefficient for sparse graphs
- Consider a random graph of n vertices, where each possible edge is included in the graph with probability 0.5
    - Based on the information theoretical lower bound, any compression scheme on expectation uses at least n^2 bits
    - Provably for this class of graphs the adjacency matrix representation is optimal
- For dense random graphs adjacency matrices are good



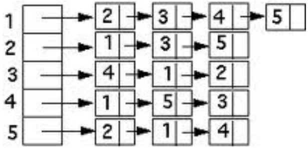
(a)

	1	2	3	4	5	6
1	0	1	0	1	0	0
2	1	0	0	1	1	0
3	0	0	0	0	1	1
4	1	1	0	0	1	0
5	0	1	1	1	0	0
6	0	0	1	0	0	1

(b)

When Are Adjacency Lists Good?

- To overcome the cost of using adjacency matrices for sparse graphs, adjacency lists are used
- Consider a random graph of n vertexes, where each vertex has only one outgoing edge and the destination is picked uniformly at random
    - Any compression scheme in expectation uses at least $n \log n$ bits
    - Provably adjacency list is optimal
- For sparse random graphs adjacency lists are good

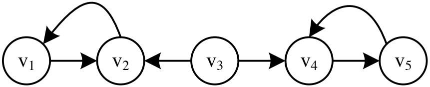


Critical Ideas

- Social networks are locally dense and globally sparse — an important, well accepted observation
- Is it possible to combine the adjacency matrix method and the adjacency list method effectively to get a better compression method?
- Critical idea: for "local" edges, use adjacency matries; for "global" edges, use adjacency lists (i.e., pointers)

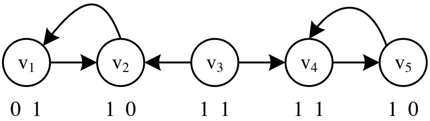
Graph Linearization

- Arrange all vertices into a sequence
- Example



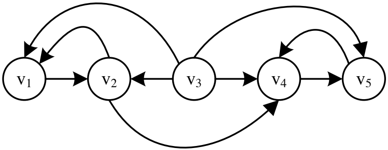
Graph Linearization

- Arrange all vertices into a sequence
- Example



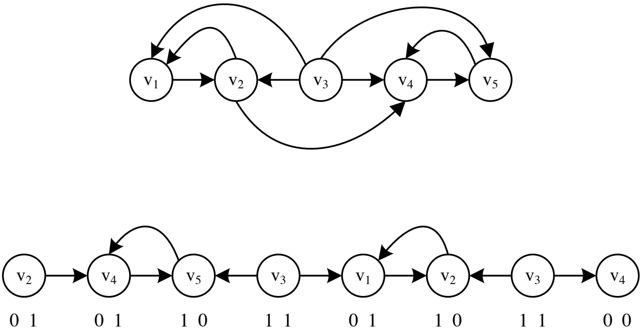
- All edges are "local" – every edge is connecting two vertices next to each other

Multi-Position Linearization



- If every vertex can only appear once, the vertices cannot be linearized such that every edge is “local”

Multi-Position Linearization

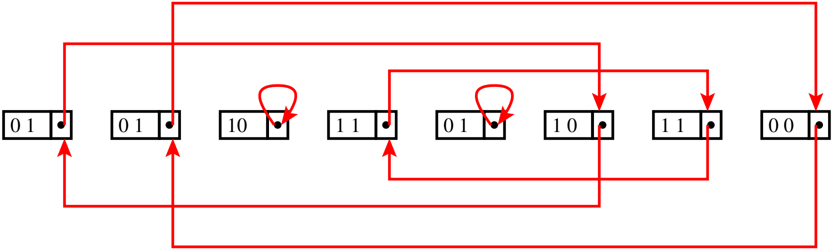


- If every vertex can only appear once, the vertices cannot be linearized such that every edge is "local"
- Multi-position linearization: a node can appear multiple times

Outline

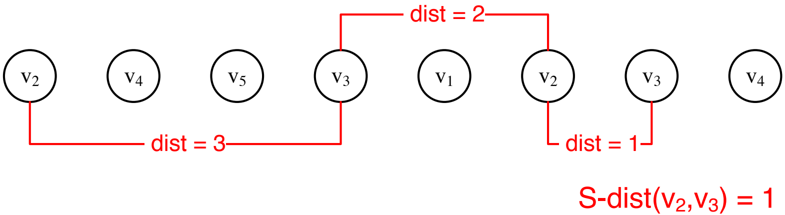
Data Structure

- An array where each cell consists of a pointer and two bits
- The index of the first appearance of a node is its ID
- We can extend the idea by using $2k$ bits for each position to encode the outlinks that are at most k positions away



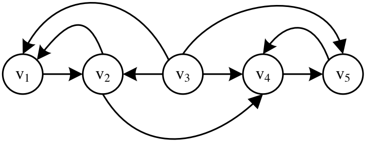
S-distance

Given a sequence S of nodes of the graph, the *S-distance* between u and v is the minimum norm-1 distance among all pairs of appearances of u and v



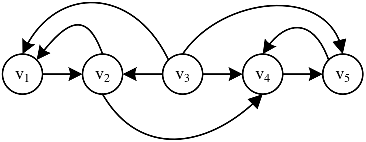
MP_k linearization

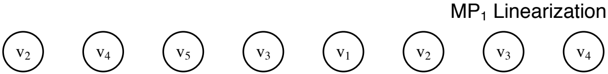
- An *MP_k linearization* of graph G is a sequence S of vertices, such that for all $(u, v) \in E(G)$, $S\text{-dist}(u, v) \leq k$



MP_k linearization

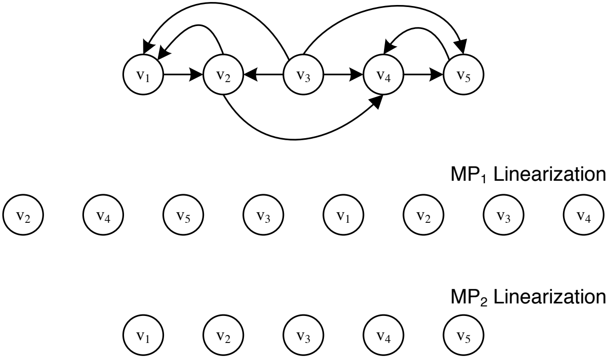
- An *MP_k linearization* of graph G is a sequence S of vertices, such that for all $(u, v) \in E(G)$, S-dist$(u, v) \leq k$





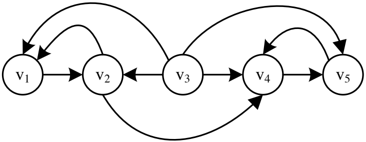
MP_k linearization

- An *MP_k linearization* of graph G is a sequence S of vertices, such that for all $(u, v) \in E(G)$, $S\text{-dist}(u, v) \leq k$

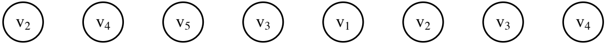


MP_k linearization

- An *MP_k linearization* of graph G is a sequence S of vertices, such that for all $(u, v) \in E(G)$, S-dist$(u, v) \leq k$
- Given MP_k linearization L of G, one can encode G using $(2k + \lceil log|L|\rceil) \times |L|$ bits, where $|L|$ is the length of L



MP1 Linearization



MP2 Linearization

Some Observations

- For a directed graph G, let $\bar{G}$ be the underlying undirected graph of G
- If $\bar{G}$ is an Euler graph, the Euler path achieves the optimal MP_1 linearization of G
  - Every edge in $\bar{G}$ appears only once, and thus the length of the vertex sequence is minimized
  - All edges in G are coded
- If $\bar{G}$ is not an Euler graph, but by adding one edge the graph becomes an Euler graph, then the Euler path of the enhanced graph (i.e., the graph with an added edge) achieves the optimal MP_1 linearization of G
  - Still, every edge in $\bar{G}$ appears only once
- In general, an extra pair of odd degree vertices in $\bar{G}$ needs one edge to make an Euler path
- Use Hierholzer's algorithm to find Euler paths in linear time

Minimum MP_1 Linearization Algorithm

Input: an underlying undirected graph $\bar{G}(V, E)$ of a directed graph G
Output: the minimum MP_1 linearization of G

$i \leftarrow 0$
while $E \neq \emptyset$ **do**
  pick a vertex v with odd degree, if there is no such a vertex, pick an arbitrary vertex with nonzero degree
  repeat
    choose an edge $(v, u) \in E$ whose deletion does not disconnect the graph, if there is no such a choice, choose an arbitrary $(v, u) \in E$
    $L[i] \leftarrow v$, $i \leftarrow i + 1$
    $E \leftarrow E - \{(u, v)\}$
    $v \leftarrow u$
  until the degree of v is 0
end while
return L

Analysis

- The algorithm partitions the edges to exactly $\frac{N_{odd}}{2}$ edge-disjoint paths, where N_{odd} is the number of vertices with odd degree (assuming $N_{odd} > 0$)
- The length of an optimal MP_1 linearization is for $N_{odd} > 0$

$$\|E\| + \frac{N_{odd}}{2}$$

- The time complexity: $O(\|E\|)$

Compression Rate: An Upper Bound

- Using MP_1 linearization to encode a graph G the bits/edge rate is at most
$$(1+\frac{1}{\bar{d}})\Big(\lceil \log_2(|V(G)|)+\log_2(\bar{d}+1)\rceil+1\Big)$$
where $\bar{d}$ is the average degree in $\bar{G}$, the underlying undirected graph of G
- The in-neighbor and out-neighbor query processing time on vertext v is
$$O\Big(\sum_{u\in N_v} deg(u)\log|V(G)|\Big)$$
- The trivial encoding of the graph that answers both in-neighbor and out-neighbor queries uses $2\log|V|$ bits/edge

From MP_1 to MP_2 Linearization

- What is the complexity of computing an optimal MP_2 linearization? An open question!
- Minimum MP_k linearization when k is part of the input is a generalization of Min-Bandwidth problem and therefore it is NP-hard
  - Min-Bandwidth problem: Find an arrangement of vertices of the graph that minimizes the maximum stretch of an edge

A Greedy Algorithm

while $E \neq \emptyset$ **do**
    find the vertex u that has the largest number of edges to the last k vertices in the current list
    remove the edges between u and the last k vertices in the list
    add u into the list
end while

- The graph gets sparser and sparser as we are removing the edges
- We use a threshold to reduce the value of k in the process of linearization

Data Sets

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

$Acc(G)$: the average clustering coefficient

$Gcc(G)$: the global clustering coefficient

$Fre(G)$: the fraction of reciprocal edges in $E(G)$

Compression Rates

(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

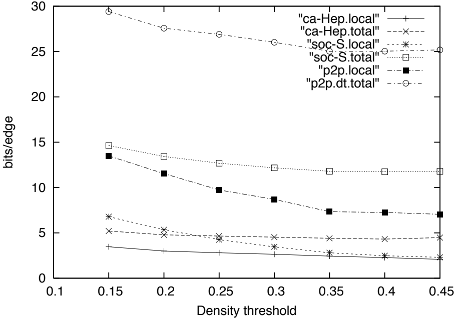
The worse cases happen on those data sets that have very poor locality measures (*Gcc* and *Fre*)

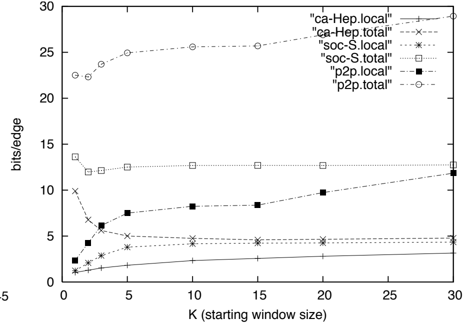
Query Processing Time

	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

- Our method spends up to 3 times more time to answer an adjacency query than that on the original graph.
- For neighbor queries, the query answering time depends on the efficiency of the linearization. The more replicates, the longer the query answering time.

Sensitivity to Parameters





Comparison with [Chierichetti *et al.* KDD'09]

- The compression rate of the method in [Chierichetti *et al.* KDD'09] on LiveJournal dataset is 14.38, but it can only answer out-neighbor queries
- Our compression rate is 13.91
- Our method can answer both in-neighbor and out-neighbor queries

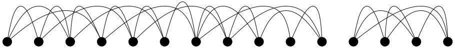
Outline

Why Lossy Compression?

- Practical aspect: by reducing noise, lossy compression may serve as a preprocessing step in social network analysis
- Theoretical aspect: using the priority of including an edge/vertex into a lossy compression, we may discover importance of edges and vertices, and identify noise edges and vertices

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 $(x, y) \in E(G_s)$, $|\phi(x) - \phi(y)| \leq k$. 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)

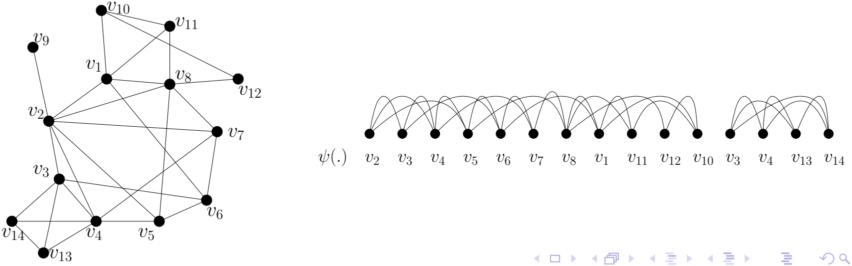


A $(3, 15)$-sequence graph

- In general, a (k, l)-sequence graph G_s may have more than one bijection between $V(G_s)$ and integers $\{1, \ldots, l\}$
- A sequence graph can be used to linearize a graph in a lossless or lossy way

Graph Linearization

- A (k,l)-sequence graph G_s is a (k,l)-**linearization** of a graph G if there exists a function $\psi : V(G_s) \to V(G)$ 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 $(\psi(x),\psi(y)) = (\psi(x'),\psi(y'))$
- G_s is a **lossless linearization** 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



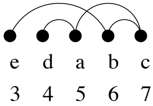
Objective Function Design

- 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
- 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
- Since communities are the essential building blocks of social networks, we advocate 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

A Simple Objective Function

- 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

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



For path $p = (d, a, c, b, e)$, the span is $7 - 3 = 4$.

- First idea: $k = l = |V(G)|$, and shorten the sum of spans of all edges

$$f_1(G_s) = \sum_{(x,y) \in E(G_s)} \alpha^{span(x,y)}, \text{ where } (0 < \alpha < 1)$$

- Shorter spans of edges in the sequence graph $\rightarrow$ higher utility
- More edges included in the compression $\rightarrow$ higher utility

Generalization

- How are paths of a certain length 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, 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
- Let $P_m(G_s)$ be the set of paths of length m in a sequence graph G_s. We extend utility function f_1 to

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

Tackling the Case of $m = 2$

- We tackle the simplest nontrivial setting $m = 2$ as the first step
  - Interestingly, several recent studies suggested that even considering random walks of short length can generate high quality results in network analysis
  - For $m \geq 3$, the problem is computationally more expensive, and 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:

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

- Our problem is highly related to (and more complex than) a family of *graph layout problems*, 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, even no constant factor approximation algorithm for any variation of these problems is known

Bounding the Objective Function

- Let G_s be a sequence graph. Then,

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

- $\alpha^{\frac{1}{2}}$ and α are constants. Heuristically, if we can obtain a sequence graph optimizing the lower bound, 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,

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

- We optimize the lower bound if we optimize

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

Connection between Parameters α and k

- 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
- For a given α, the maximum span of all edges in the optimal sequence graph is at most $\log_\alpha \frac{\alpha(1-\alpha)}{4}$
- We use $k = \log_\alpha \frac{\alpha(1-\alpha)}{4}$ to estimate α
  - 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 $k/2$.

Greedy Heuristic Search

- We initialize G_s with a random ordering of the vertices of G. There is no edge in G_s at this stage
- Iteratively we consider all vertices for possible reallocation
  - Find 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
- Details in our paper

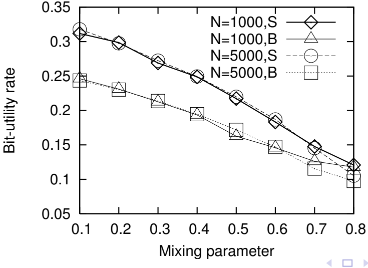
Pseudocode (the Framework)

Input: G: input network, k: local range, l: length of compression ($l \geq |V(G)|$)
Output: $SeqG$: sequenced compression

Initialize $SeqG$ with a random ordering of nodes, $\alpha \leftarrow EstimateAlpha(k)$
repeat
  $b \leftarrow f(SeqG, \alpha)$
  for all $u \in V(G)$ **do**
    $IPos \leftarrow NULL$, $DPos \leftarrow NULL$, $(IPos, Nbh) \leftarrow ReAllocate(u, G, SeqG, \alpha)$
    if $(IPos \neq NULL)$ and $(Length(SeqG) = l)$ **then**
      $DPos \leftarrow SeqG.LowestBenf(IPos - k, IPos + k)$
    end if
    $x \leftarrow UtilityIncrease(IPos, Nbh, SeqG)$, $y \leftarrow UtilityDecrease(DPos, SeqG)$
    if $x - y > 0$ **then**
      $Insert(IPos, Nbh, SeqG)$, $Delete(DPos, SeqG)$
    end if
  end for
  $\alpha \leftarrow f(SeqG, \alpha)$
until convergence condition

Evaluation Methodology

- Compression rate does not have a straightforward meaning in lossy compression
- The **bit-utility rate** is the ratio of the number of edges encoded in the lossy compression over the total number of bits
- To generate synthetic data sets, we use the LFR benchmark, and the same settings as those used by Fortunato



Evaluating Community Preservation Using Proximity Graph

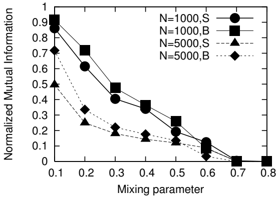
- Let G_s be a linearization of G, and $\psi : V(G_s) \to V(G)$ the mapping. Note that $V(G_s) = \{1, \cdots, |V(G_s)|\}$. The **proximity graph** of G with respect to G_s is defined as follows. Consider $(u, v) \in E(G)$ and $(i, j) \in E(G_s)$ such that $|i - j| \leq k$ and $\psi(i) = u$, $\psi(j)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

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

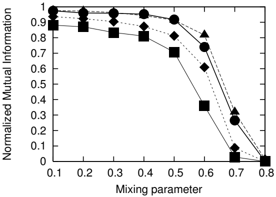
- The first (second) term is over all the edges associated with position i (j) that pass over position j (i). If there is no such a pair of (i, j), then the weight for (u, v) is 0. If there are more than one such pair, for each of those pairs, we compute the weight and take the sum over all of them
-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

Comparing Normalized Mutual Information on Original Graph and Proximity Graph

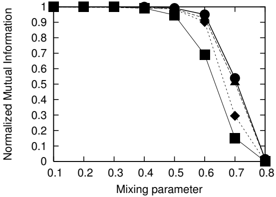
- Use the community finding algorithm by Clauset *et al.*
- Use several (5 in our experiments) independent linearizations to obtain an aggregated proximity graph



Original networks

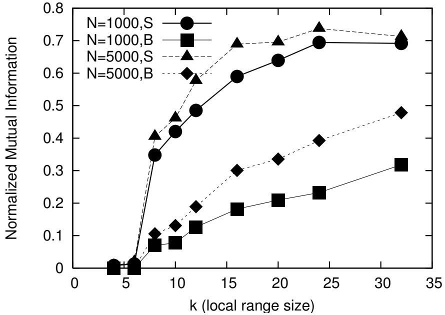


Proximity networks

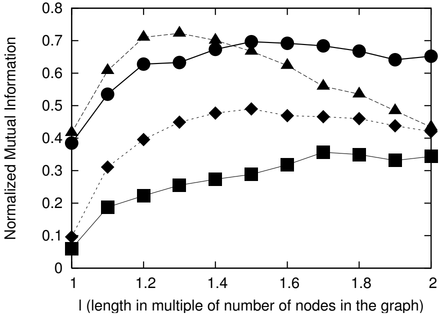


Aggregated proximity networks

Effect of Local Range Size k and Length of Sequences l



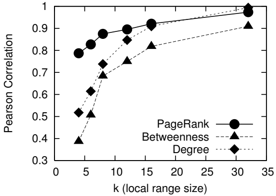
$\mu = 0.6$ and $L = 1.2 \times N$



$\mu = 0.6$ and $K = 8$

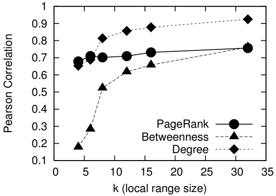
Evaluation Using Centrality

- Use betweenness, PageRank, and degree
- The centrality of all vertices forms a vector
- Evaluate the Pearson Correlation on centrality vectors



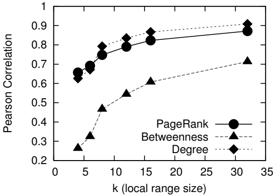
Collaboration network

5,242 nodes, 14,990 edges



Wiki vote network

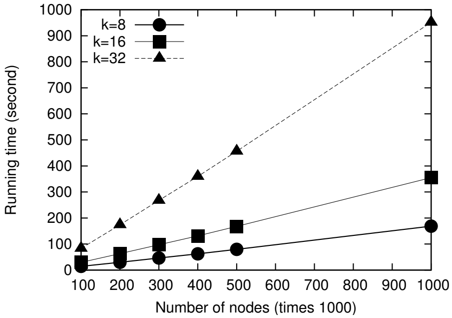
1,133 nodes, 5,451 edges



Email exchange network

7,115 nodes, 100,763 edges

Single Iteration Runtime



Outline

Conclusions

- Online social networks are more and more popular, and larger and larger
- We developed a novel framework for social network representation which leads to compression
  - Our method does not use any coding yet
  - Compression rate guaranteed in MP_1
  - Our method has a comparable compression rate than the state-of-the-art method
  - Our method are query friendly
- Importantly, our method reduces the problem of compressing a graph to an intuitive combinatorial problem, and lead to new understanding of structural properties in social networks

Future Work

- Complexity or NP-hardness for MP_k when k is fixed
- Better heuristic algorithms for MP_k linearization (if it is NP-hard)
- Plugging in other compression techniques into our framework
- Using graph linearization to gain better understanding of graph properties

Our Papers

- Hossein Maserrat and Jian Pei: Neighbor query friendly compression of social networks. In *KDD* 2010: 533-542
- Hossein Maserrat and Jian Pei: Community preserving lossy compression of social networks. In *ICDM* 2012