Neighbor Query Friendly Compression of Social Networks

Hossein Maserrat [1] Jian Pei [1]

[1]School of Computing Science
Simon Fraser University
{hmaserra, jpei}@cs.sfu.ca

Outline

1. Background
   - Motivations
   - Existing Approaches
2. Our Framework
   - Some Observations
   - The Idea
3. Formalization
   - Concepts
   - Theoretical Results
4. Experiments
   - Heuristic
   - Results

Motivations

- Real world networks (i.e. webgraphs, online social networks) are huge
- Even though there are distributed platforms for large scale graph processing still graph compression is beneficial:
  - Processing larger portion of graph per computer
  - Reducing the communication cost
- Ideally we like to be able to run different types of query on the compressed graph

Existing Approaches: Web graphs

Boldi and Vigna [WWW04]

- Locality of webgraphs: A large percentage of links are intra-domain
- Sort by URL to improve locality of links
- Sort the list of out-links for each node
- Use ζ codes to encode the gaps between out-links
- This compress the webgraphs down to almost 2 bits per edge

Existing approaches: Friendship networks

Chierichetti et. al. [KDD09]

- There is no natural ordering of vertices for friendship networks
- Instead of sorting by URL in the previous approach, use shingle ordering
- 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 is not nearly as good as webgraphs

Some Observations

Observation: The optimality of adjacency matrix representation

- Consider the class of random graph on n vertices where each possible edge is included in the graph with probability half
- Based on information theoretical lower bound, Any compression scheme on expectation uses at least n^2 bits
- Provably for this class of graphs adjacency matrix is the optimal schema
- Message: Roughly speaking for dense random graph adjacency matrix is the best option

Some Observations

Observation: The optimality of adjacency list representation:

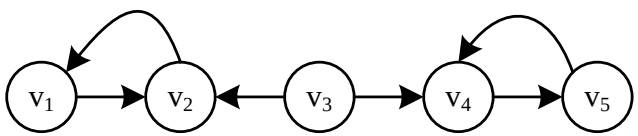
- Consider the class of random graph on n vertexes, where:
  - Each vertex has only one outgoing edge
  - The destination of that edge is picked uniformly at random
- For this class of graphs any compression scheme on expectation uses at least $n \log n$ bits
- In this case provably adjacency list is the optimal schema
- Message: Roughly speaking for sparse random graph adjacency list is the best option

The Idea

- It is well known that social networks are locally dense and globally sparse
- Question: Is it possible to combine the adjacency matrix and adjacency list effectively to get a compression schema?
- General Idea: For "local" edges use adjacency matrix and for "global" connections use pointers

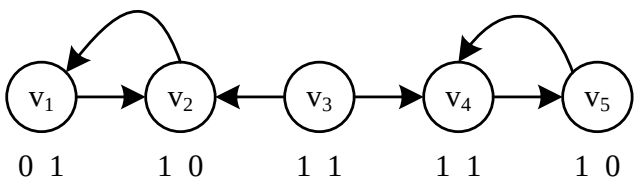
The Idea

- Let's consider a very simple case
- Assume a linear arrangement of nodes is given
- Notice that all the edges are "local" (i.e. every edge is connecting two nodes next to each other)



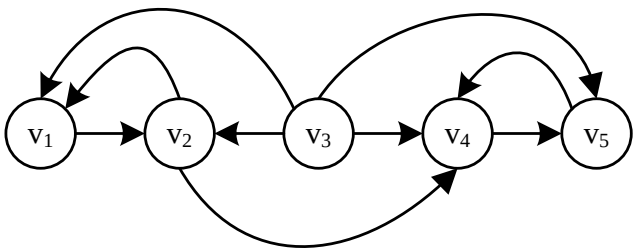
The Idea

- Let's consider a very simple case
- Assume a linear arrangement of nodes is given
- Notice that all the edges are "local" (i.e. every edge is connecting two nodes next to each other)



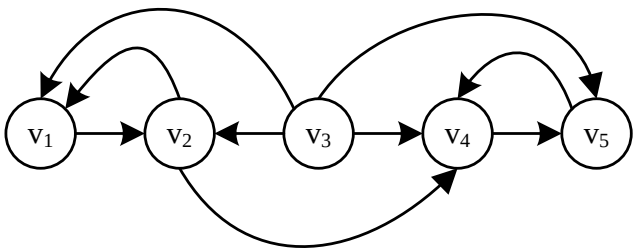
The Idea

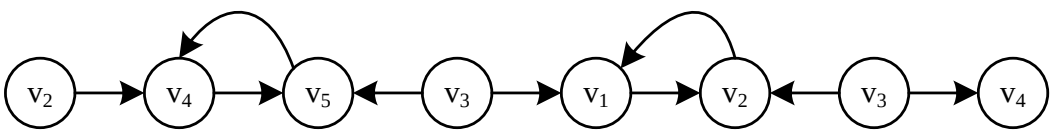
- Now we consider a more sophisticated case
- In this case there is no arrangement of vertices such that every edge is "local"
- Relaxation: A node can appear more than once



The Idea

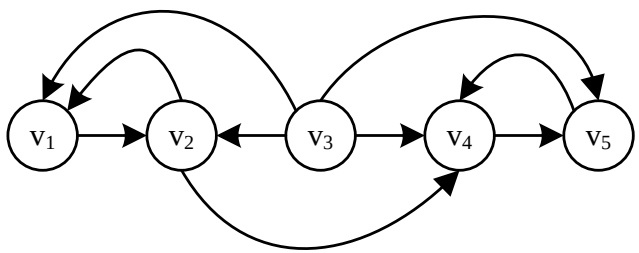
- Now we consider a more sophisticated case
- In this case there is no arrangement of vertices such that every edge is "local"
- Relaxation: A node can appear more than once

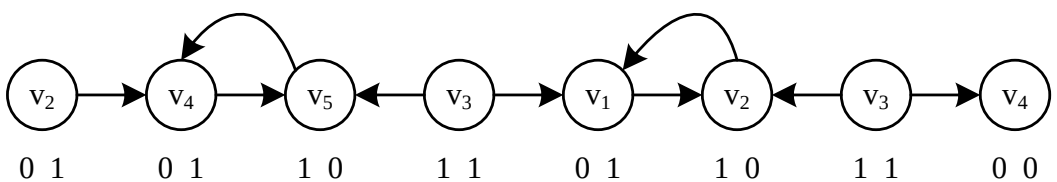




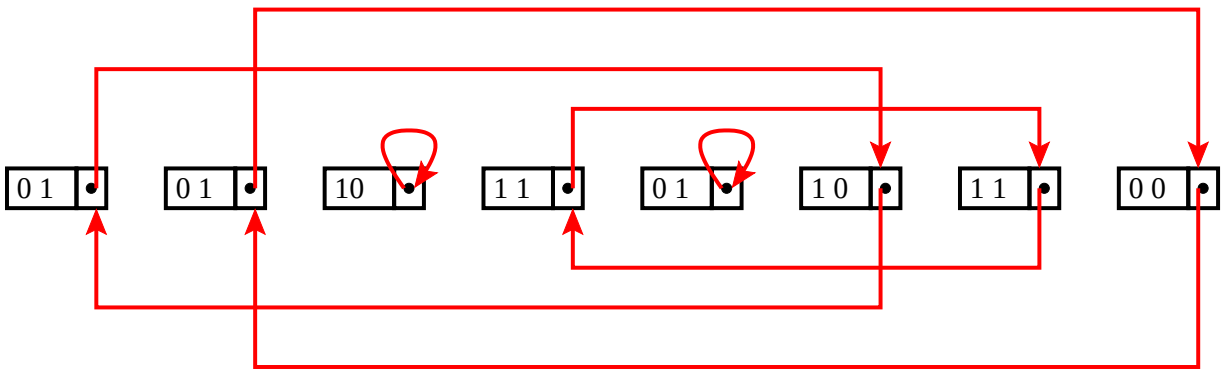
The Idea

- Now we consider a more sophisticated case
- In this case there is no arrangement of vertices such that every edge is "local"
- Relaxation: A node can appear more than once





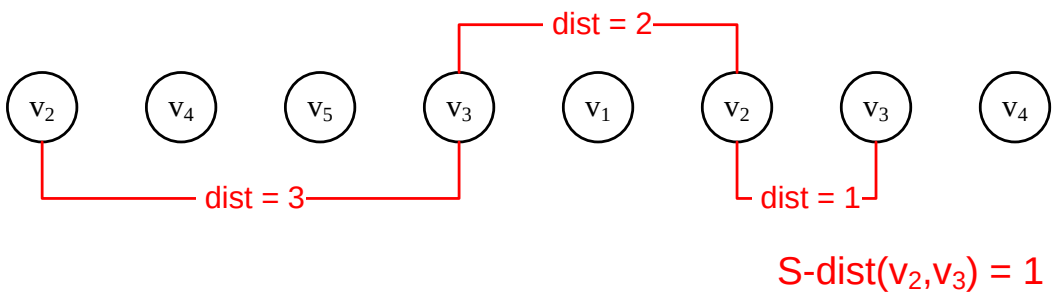
Representation Schema



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

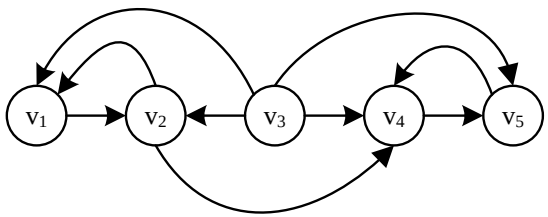
S-distance

- 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*



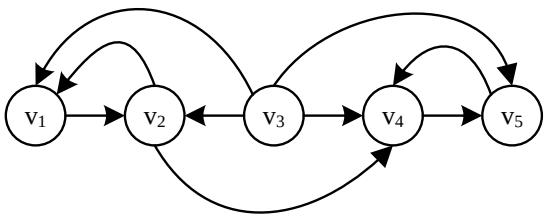
MP_k linearization

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

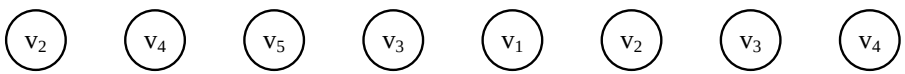


MP_k linearization

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

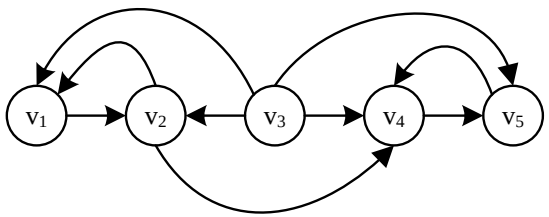


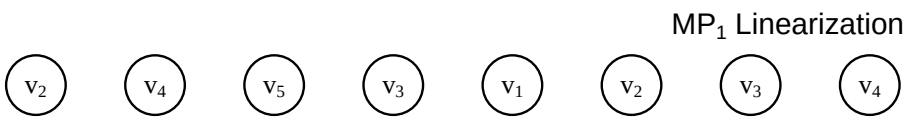
MP_1 Linearization

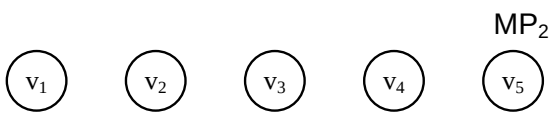


MP_k linearization

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

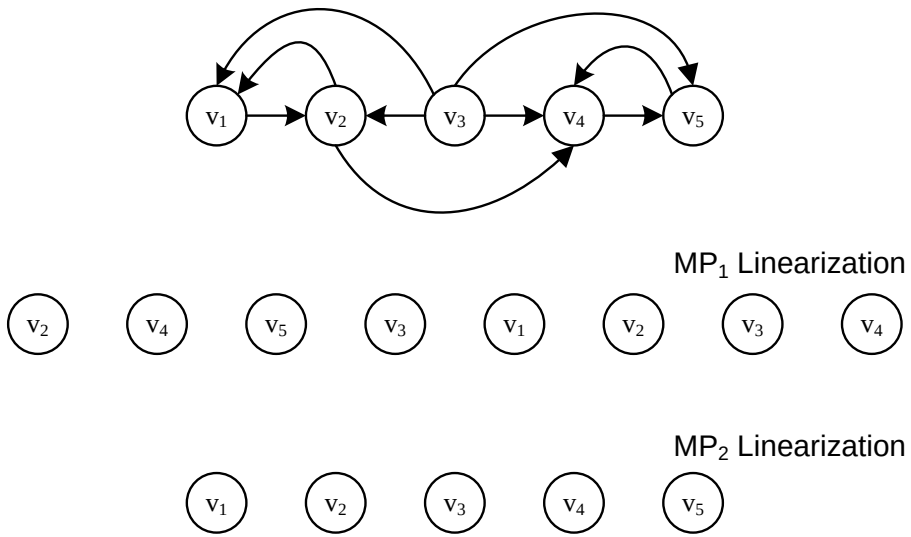






MP_k linearization

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

Finding optimal MP_1 linearization

Minimum MP_1 linearization

1. Start from a node with odd degree, if there is no such a node, start from an arbitrary node with nonzero degree
2. Choose an edge whose deletion does not disconnect the graph, unless there is no other choice
3. Move across the edge and remove it
4. Keep removing edges until getting to a node that does not have any remaining edge to choose
5. If the graph is not empty go to step 1

- This algorithm partitions the edges to exactly $N_{odd}/2$ 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 $|E| + N_{odd}/2$ ($N_{odd} > 0$)
- It can be implemented in $O(|E|)$ time

Compression Rate: Upper Bound

Using MP_1 linearization to encode a graph G the bits/edge rate is at most

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

while the in-neighbor and out-neighbor query processing time 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

Hardness Results

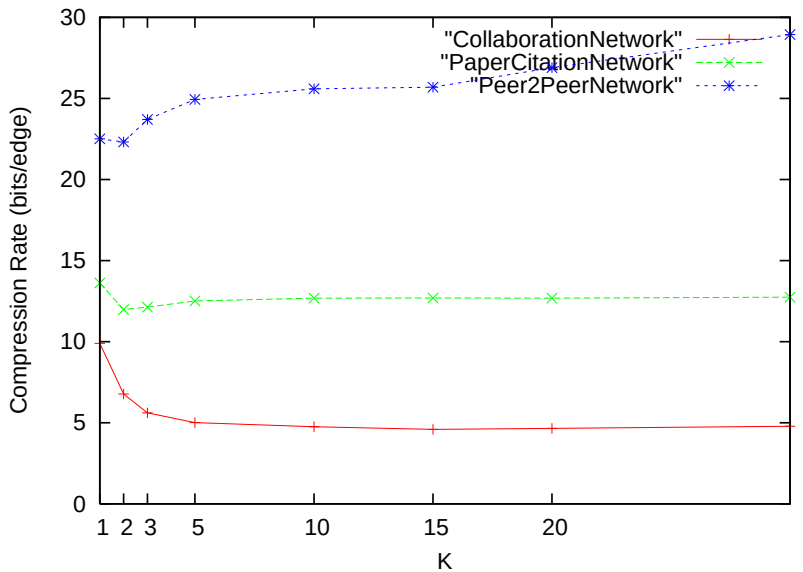
- How hard is it to compute an optimal MP_2 linearization?
  - We don't know, pretty hard I guess!
- Minimum MP_k linearization when k is part of the input is a generalization of Min-Bandwidth problem and therefore it is NP-hard
- Min-Bandwidth problem: Find an arrangement of vertices of the graph that minimize the maximum stretch of an edge

MP_k linearization: Heuristic

- A greedy heuristic to compute MP_k linearization:
  1. Start with a random node and add it to list
  2. Find the node that has the most number of edges to the last k nodes in the list
  3. Remove the edges between this node and the last k nodes in the list
  4. Add the node to the list
  5. Repeat until no edge is left
- The graph gets sparser and sparser while we are removing the edges
- We use a threshold to reduce the value of k in the process of linearization

Experimental Results

	Dataset statistics			Query processing time (ns)			
				Adj query		Neigh. query	
	$\|V\|$	$\|E\|$	FCT	Comp.	Adj. list	Comp.	Adj. list
CN	12006	236978	0.659	520	400	1849	19
PCN	34546	421534	0.145	1300	480	2745	28
P2PN	26518	65369	0.004	500	320	1488	50
LJN	4845609	68475391	0.288	3050	1130	9734	49



Comparison

Comparison: The compression rate of the previous schema [KDD09] on LiveJournal dataset is 14.38 while it can only answer out-neighbor queries, our compression rate is 13.91 while our method can answer both in-neighbor and out-neighbor queries

Conclusions

- We introduce a novel framework for representing graphs
- In its simplest settings, our framework comes with an upper bound on bits/edge rate
- Our method can efficiently answer more types of query and retain the comparable compression rate than the state-of-the-art methods
- Our framework reduces the problem of compressing a graph to an intuitive combinatorial problem

Future works

- Finding smarter algorithms for linearizing a graph
- Using other compression techniques on top of our framework
- Hardness result for MP_k linearization when k is fixed