On Compressing Time-Evolving Networks

Sudhindra Gopal Krishna, Sridhar Radhakrishnan, Michael A. Nelson, Amlan Chatterjee, and Chandra N. Sekharan

Presenter: Sudhindra Gopal Krishna

sudhi@ou.edu

School of Computer Science, The University of Oklahoma, Norman, OK

Presenter’s Bio

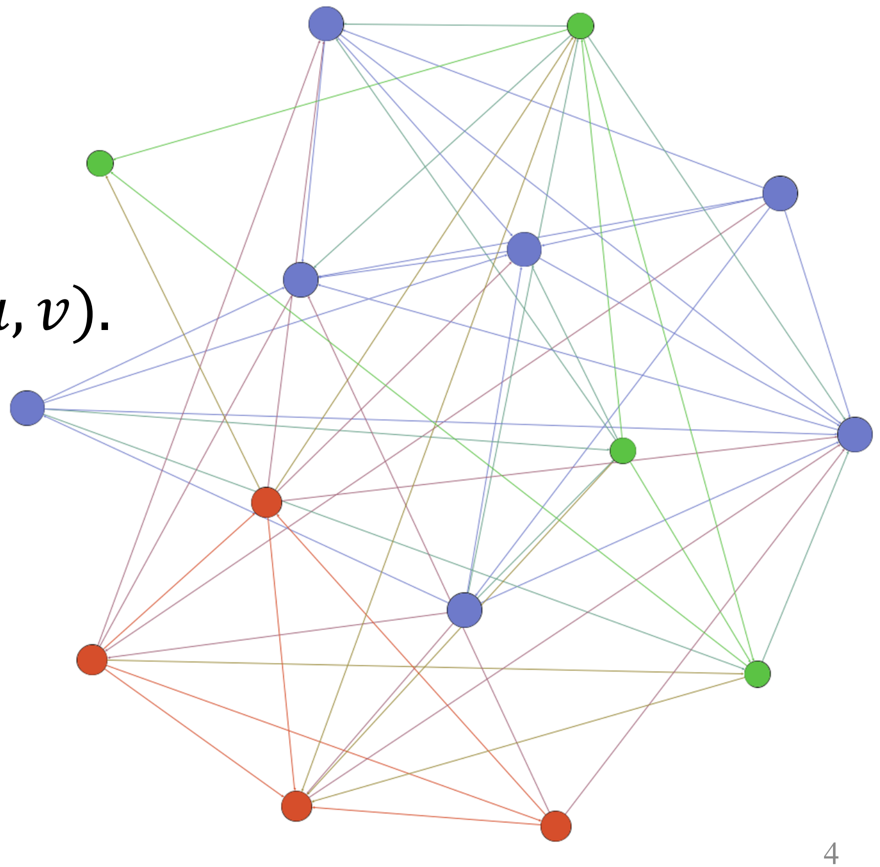
- Sudhindra Gopal Krishna is a Ph.D. Candidate in the School of Computer Science at the University of Oklahoma.
- Sudhindra graduated with a Master's Degree in Computer Science in Spring 2017.
- Current research interests:
  - Algorithms, Data Compression, Graphs, Network Science.

Contents

- Introduction
- Background
- Graph Representation
- Research Methodology
- Experimental Results
- Conclusion
- Future Work

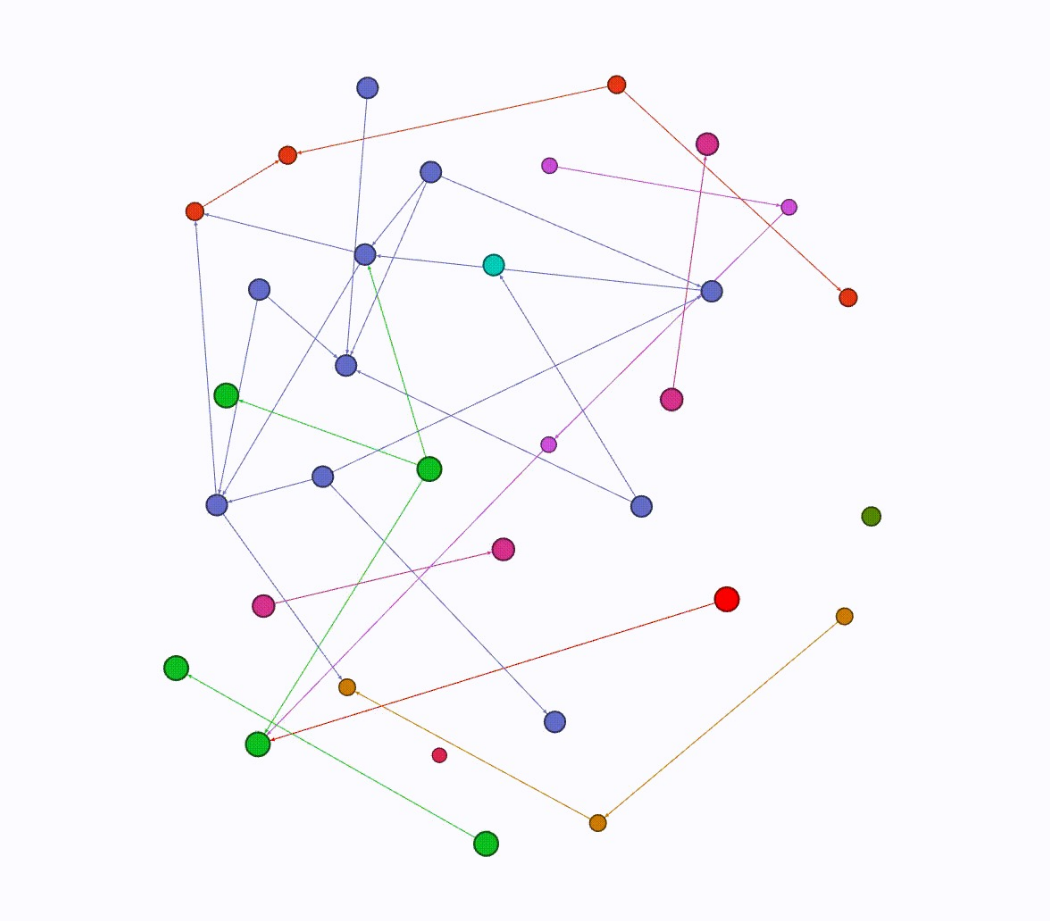
Introduction

- A Graph (aka Network) is a structure represented as a set of nodes (or vertices) connected by a set of links (or edges).
- A Graph is formally represented as $\mathrm{G} = (V, E)$.
  - V represents a non-empty set of vertices, and
  - E represents a set of edges.
- Each edge in the graph is represented as a tuple (u, v).



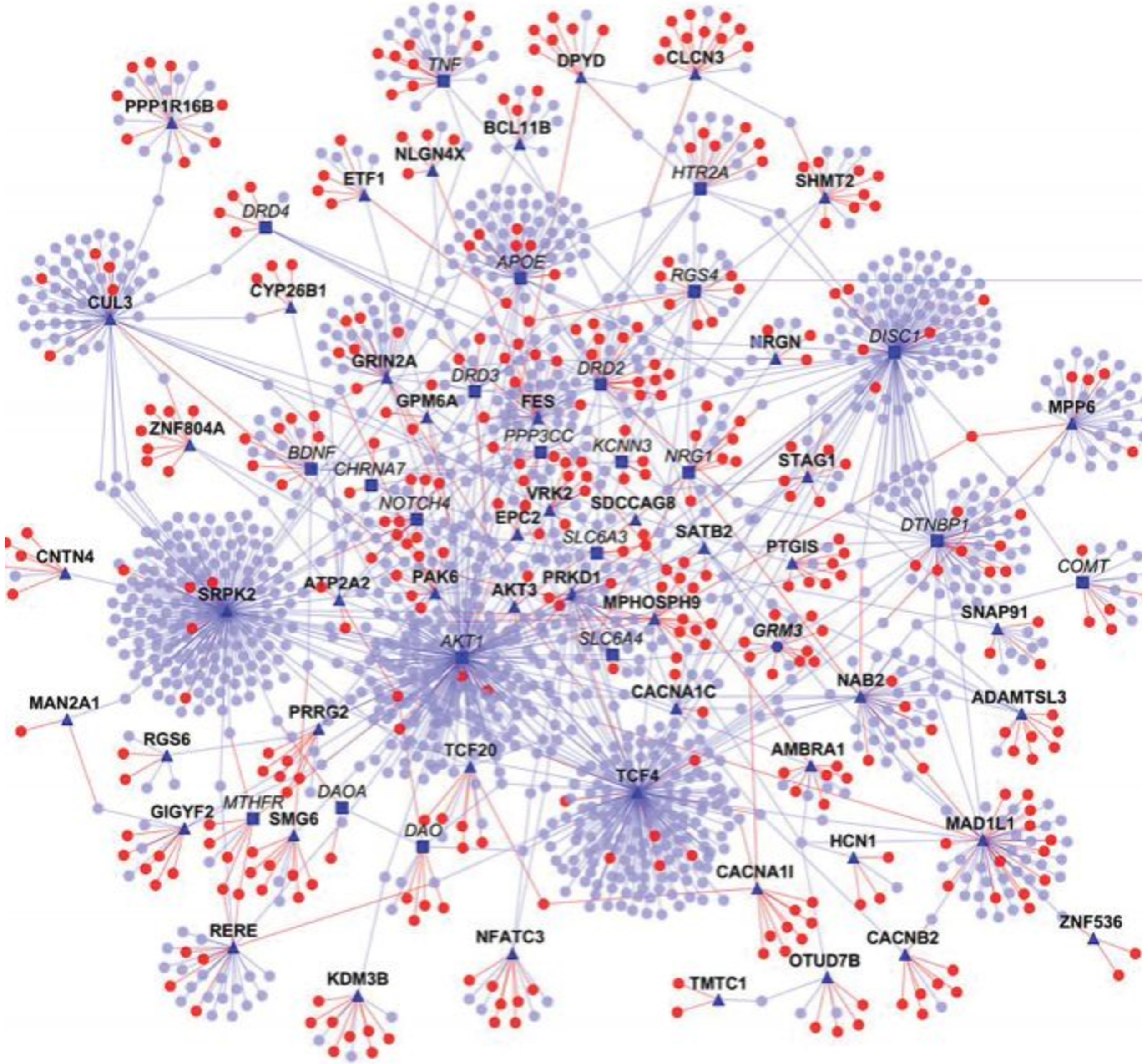
Time-Evolving Graphs

- Time-evolving graphs can be stored as an extension to the static graphs.
  - By adding the third and the fourth dimensions, which are the start time (t_i), and the end time (t_j).
  - Formal notation of a time-evolving graph is $G_i = (V_i, E_i, \tau)$,
    - where i is a time-frame, and
    - τ is a time interval (t_i, t_j).
  - So, a link in a time-evolving graph is a 4-tuple (u, v, t_i, t_j).
- This is often seen in social networks.



Massive Graphs

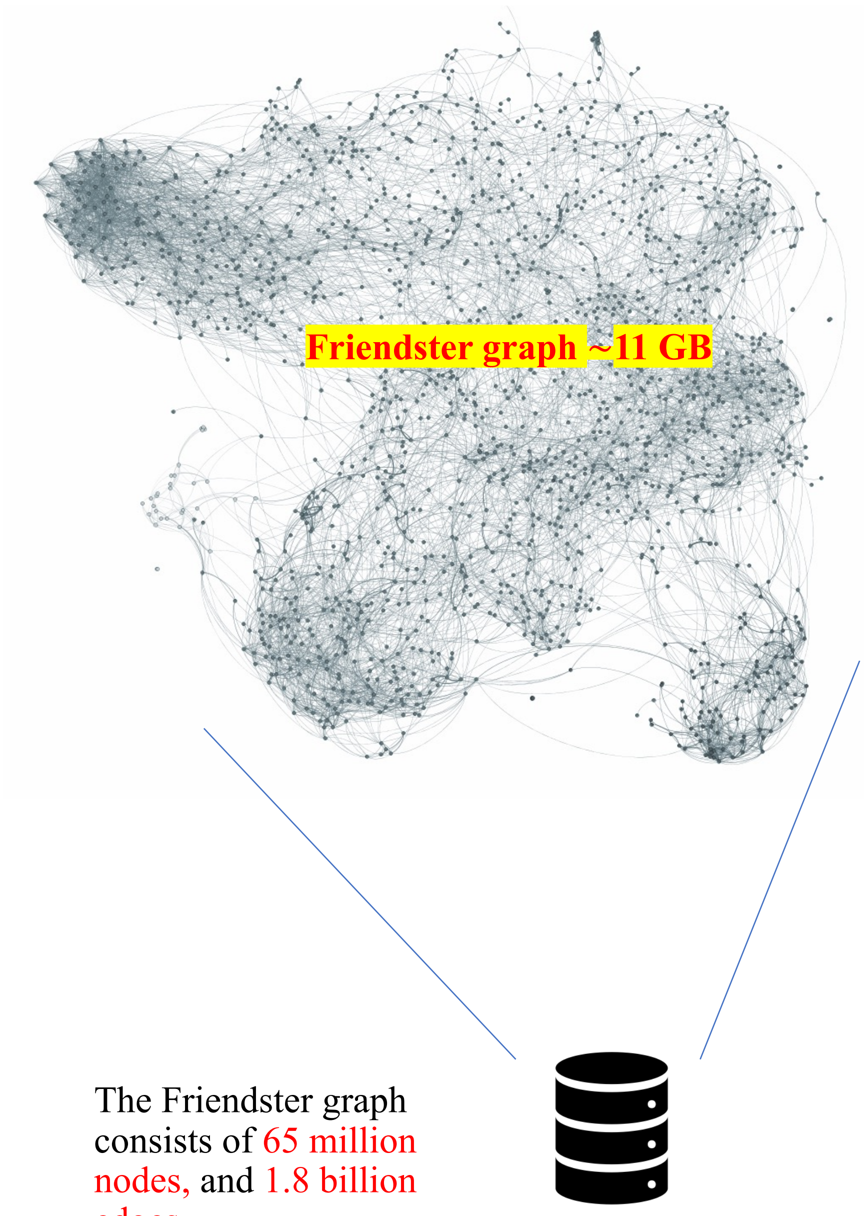
- The graphs that do not fit into the memory for analysis are referred to as massive graphs.
- In reality, these graphs are very sparse.
- Where **Sparsity** is a measure of how many links exist out of all possible links in the graph.



https://www.genengnews.com/insights/protein-protein-interactions-get-a-new-groove-on/

Storage Solution

- One way to address the issue of storing massive graph is to compress the data.
- There are two ways to compress a graph
  - Graph (Matrix) based compression, and
  - Node (Row-by-Row) based compression.
- All the algorithms in this paper are based on row-by-row compression.



The Friendster graph consists of 65 million nodes, and 1.8 billion edges.

RAM 8 GB

https://colah.github.io/posts/2014-02-Fanfiction-Graphs-PageRank/img/graph-HP-blank.png

Background – Node Based Compression

- Time-evolving graphs can also be stored as a series of static graphs called snapshots.
- Row-by-Row compression for static graphs,
  - Compressed Sparse Row (CSR) [1] was introduced in 1976, is one of the most common data structures used in representing a graph.
  - BackLinks Compression (BLC) [2] was introduced in 2009, is a modified web-graph compression method developed by Boldi and Vigna.
  - Compressed Binary Tree (CBT) [3] was introduced in 2017, is a state-of-the-art structure, which eliminated the need for any intermediate structure to compress the graph.
- This paper introduces CSR for time-evolving graphs, also the combinations of CSR and CBT.

[1] R. A. Snay, "Reducing the profile of sparse symmetric matrices," *Bulletin Ge´ode´sique*, vol. 50, no. 4, pp. 341–352, 1976.

[2] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan, "On compressing social networks," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Dis- covery and Data Mining*, KDD '09, (New York, NY, USA), p. 219–228, Association for Computing Machinery, 2009.

[3] M. Nelson, S. Radhakrishnan, A. Chatterjee, and C. Sekharan, "Queryable Compression on Streaming Social Networks," in *Big Data (Big Data), 2017 IEEE International Conference on*, IEEE BigData '17, IEEE Computer Society, 2017.

Background – Matrix Based Compression

- Matrix based compression for time-evolving graphs,
  - In 2016, Caro et al. developed $ck^d - tree$, by adapting the concepts of quadtree compression.
  - Evelog is a compressed adjacency log structure, based on log of events strategy.
  - Other legacy time-evolving compressions are CAS, CET, and TGCSA.
- For this paper, only $ck^d - tree$ the matrix-based compression is considered for comparison purposes, as $ck^d - tree$ has shown better results than all the other matrix-based compression.

D. Caro, M. A. Rodriguez, N. R. Brisaboa, and A. Farina, "Compressed kd-tree for temporal graphs," *Knowl. Inf. Syst.*, vol. 49, pp. 553–595, Nov. 2016.

Compressed Sparse Row (CSR)

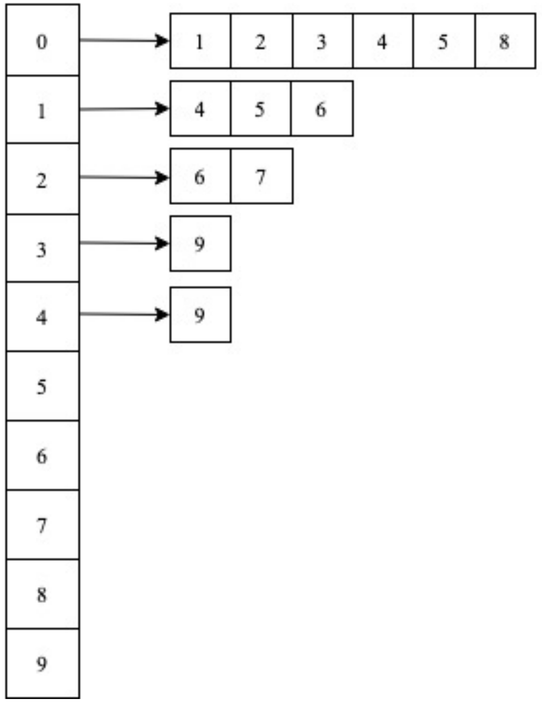
- Here is an example of a Compressed Sparse Row (CSR) structure for a static undirected graph.
- In this structure it is sufficient to store only the upper triangular part of the matrix shown in green.

	0	1	2	3	4	5	6	7	8	9
0	0	1	1	1	1	1	0	0	1	0
1	1	0	0	0	1	1	1	0	0	0
2	1	0	0	0	0	0	1	1	0	0
3	1	0	0	0	0	0	0	0	0	1
4	1	1	1	0	0	0	0	0	0	1
5	1	1	1	0	0	0	0	0	0	0
6	0	1	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	1	0	0	0	0	0	0	0	0	0
9	0	0	0	1	1	0	0	0	0	0

Degree Array

0	1	2	3	4	5	6	7	8	9
6	3	2	1	1	0	0	0	0	0

Adjacency Array



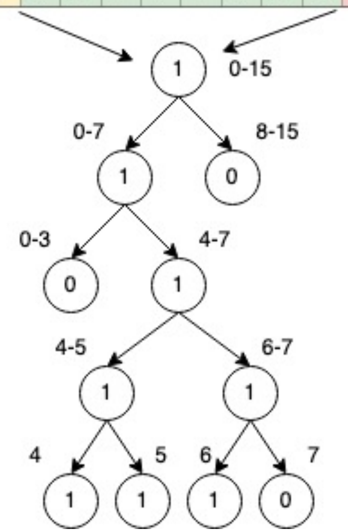
R. A. Snay, "Reducing the profile of sparse symmetric matrices," *Bulletin Ge ́ode ́sique*, vol. 50, no. 4, pp. 341–352, 1976.

Compressed Binary Tree (CBT)

- An example of a Compressed Binary Tree (CBT) structure for a static undirected graph.
- In this structure it is sufficient to store only the upper triangular part of the matrix shown in green.

	0	1	2	3	4	5	6	7	8	9
0	0	1	1	1	1	1	0	0	1	0
1	1	0	0	0	1	1	1	0	0	0
2	1	0	0	0	0	0	1	1	0	0
3	1	0	0	0	0	0	0	0	0	1
4	1	1	1	0	0	0	0	0	0	1
5	1	1	1	0	0	0	0	0	0	0
6	0	1	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	1	0	0	0	0	0	0	0	0	0
9	0	0	0	1	1	0	0	0	0	0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0

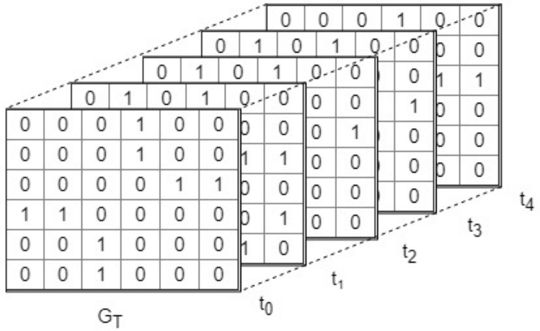


The bitString for row 2 at time T_0 is 1101111100.
The bitString for the entire graph at time T_0 is 11110111111111011000 1101111100 11010110 101001 101001 0 0 0 0 0

M. Nelson, S. Radhakrishnan, A. Chatterjee, and C. Sekharan, "Queryable Compression on Streaming Social Networks," in *Big Data (Big Data), 2017 IEEE International Conference on*, IEEE BigData '17, IEEE Computer Society, 2017.

Time-evolving graph

- A time-evolving graph can be represented as several snapshots of a static graph.

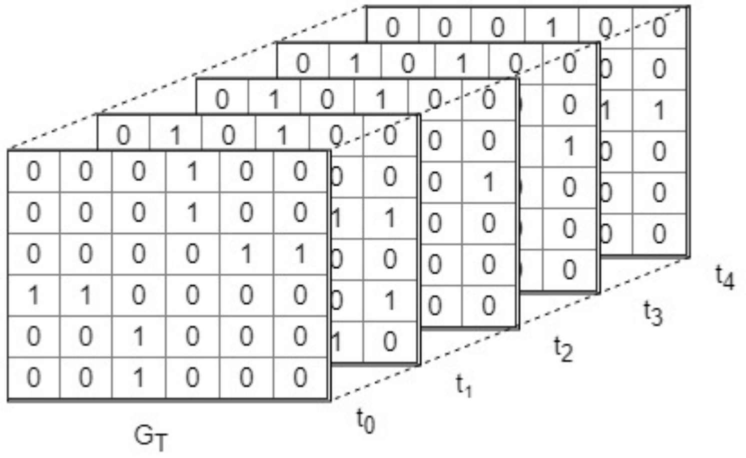


- Therefore, each node in the graph has to store two sets of information,
  - One is the neighborhood array, and
  - Second, a time array to keep track of changes between each snapshot.
- Changes at each snapshot can either be,
  - Addition of an edge, or
  - Deletion of an edge, or
  - No changes at all.

Representation of Time-Evolving Graphs

- For example,
  - Consider node 0, which is the first row in every snapshot.
  - The time-frame array is as follows:

Node 0	1	1	0	0	1



- Since the time instants are finite, for most of the examples, the range of the time-frames does not exceed more than 10,000.
- Hence, the time array is small compared to the entire graph.

Compression Technique

- In this paper, the time-array and the neighborhood-array are compressed in one of two methods.
  - Compressed Sparse Row (CSR), or
  - Compressed Binary Tree (CBT).
- Either of the two methods can be chose depending up on the size and the nature of the array.
- These arrays are encoded as a series of bits.
- For the CSR, this encoding method is referred to as bit-packing.

Bit-Packing Algorithm

- Each number in the array is represented as bits.
- Number of bits required to store each number depends on the largest number in the array.
- Then the bits are stored as an unsigned bit array.

Unsigned int	1	3	5	10	16	26
Unsigned bit	00001 00011 00101 01010 100000 11010 00					

- A memory location can hold up-to 32 bits anymore than 32, the number is split to fit the current memory location and rest of the bits are stored in the following memory location.

Unsigned int	1	3	5	10	16	26	30
Unsigned bit	00001 00011 00101 01010 10000 11010 11						
	110 00000 00000 00000 00000 00000 0000						

Input Structure

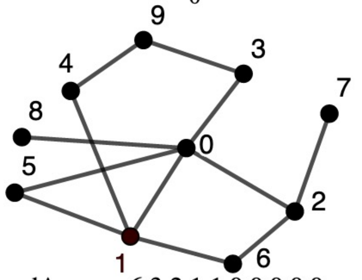
- By the definition of the time-evolving graph, each link/edge is represented as a 4-tuple (u, v, t_i, t_j).
- But the input can be reduced to a 3-tuple (u, v, t).
- Every odd occurrence of the triples indicates the link is active, and every even occurrence indicates otherwise.
- The input to the graph (edgelist) is first sorted with respect to the time-frames.
- For each time-frame, the edges are sorted with respect to the node number.

The Combinations

- Because of the two independent arrays for each node, the arrays can be stored in either be stored using CSR or CBT.
- The paper evaluates all four combinations to store the graph.
- Novel CSR – CSR: Both the time-array and the neighborhood arrays are stored using CSR.
- Novel CSR – CBT: The time-array is stored as CSR, and the neighborhood array is stored as CBT.
- Novel CBT – CSR: The time-array is stored as CBT, and the neighborhood array is stored as CSR.
- CBT – CBT: Both the time-array and the neighborhood arrays are stored using CBT.

Graphical Representation of CSR-CSR

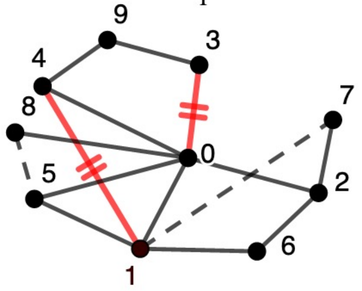
G @ T_0



dArray = 6 3 2 1 1 0 0 0 0 0

csrArray = 1 2 3 4 5 8 4 5 6 6 7 9 9

G @ T_1

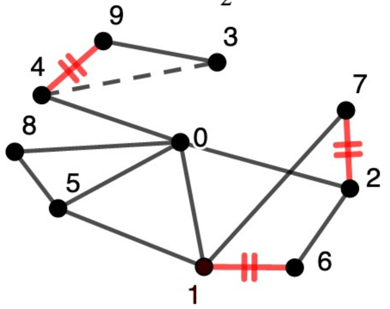


uNodes = 0 1 5

degree = 1 2 1

csrArray = 3 4 5 8

G @ T_2

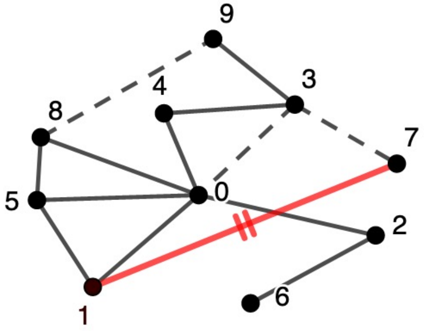


uNodes =1 2 4

degree = 1 1 2

csrArray = 3 6 7 9

G @ T_3



uNodes = 0 1 3 8

degree = 1 1 1 1

csrArray = 3 7 7 9

CSR as unsinged char:

G@T_0
01101100010010001000
100001001100001010100010010101110000100110000101010

G@T_1
000100101 100010100 110000101010001

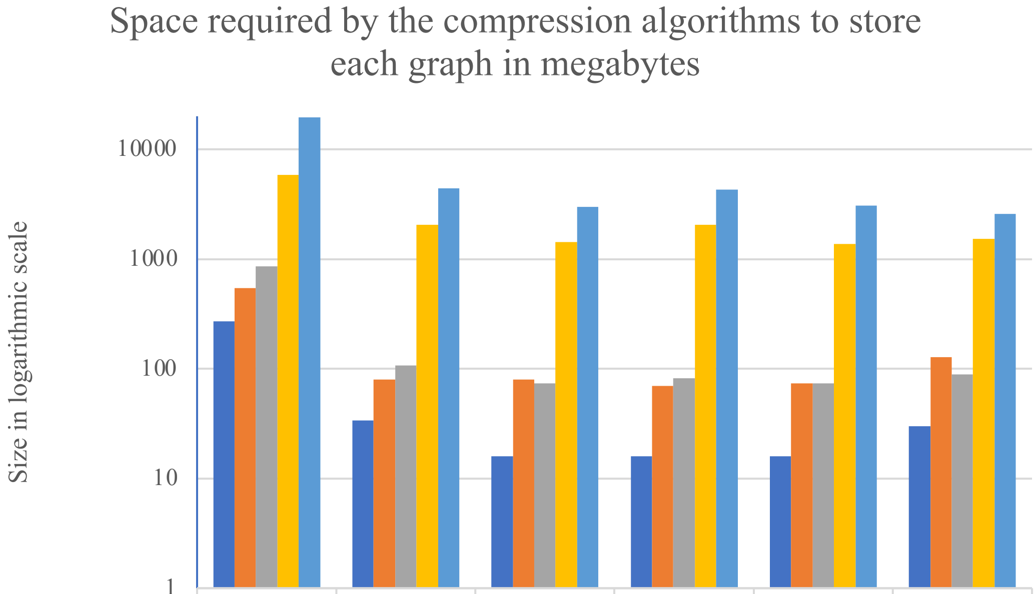
G@T_2
100010001 100100010 0010111000010101000

G@T_3
0000100011000001 1111 1100111011101001

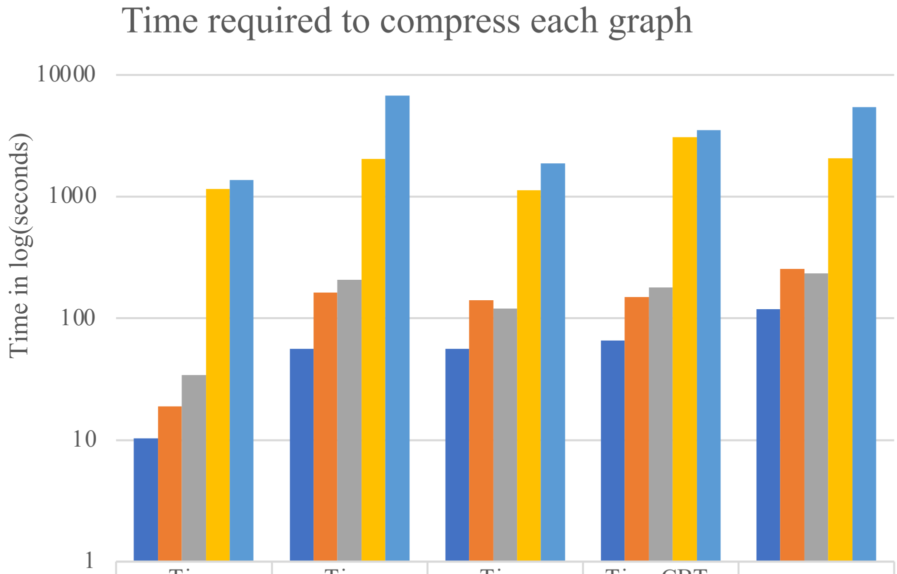
Querying operations

- Two querying operations are performed on all the algorithm on 1000 random chosen vertices.
- Given a node, and
  - A time t_i, a neighborhood query fetches all the neighbors of that node at time t_i.
  - An interval t_i to t_j (inclusive), a neighborhood query fetches all the neighbors of that node between time t_i and time t_j.
- Given nodes $(u,\ v)$, and
  - A time t_i, the edge existence method returns if the edge is active/inactive at time t_i.
  - An interval t_i to t_j (inclusive), the edge existence method returns if the edge is active/inactive between time t_i and time t_j.

Compression Result



	.txt	CSR - CSR	CSR-CBT	CBT - CSR	CBT-CBT	ck^d-tree
CommNet	271.6	34	16	16	15.9	30
PowerLaw	546.9	80	80	70	73.8	128
Flickr-Days	860	107	73.8	82	73.8	89
Wiki-edits	5836.8	2048	1433.6	2048	1372.16	1536
Yahoo-Netflow	19456	4403.2	2969.6	4300.8	3061.76	2560



	Time CSR_CSR	Time CSR_CBT	Time CBT_CSR	Time CBT-CBT	Time ckd
CommNet	10.25	56.19	55.8	65.5	119
PowerLaw	18.94	162.23	141.21	149	254
Flickr-Days	34.16	208.39	120.015	179	235
Wiki-edits	1158	2042.88	1126.85	3081	2059
Yahoo Netflow	1372	6770.71	1874.95	3506	5471

Experimental Results

Graphs	Nodes	Contacts	Time Frames	neigh_CSR_Ti	neigh_CSR_Ti_Tj	Edge exists_CSR_Ti	Edge exists_CSR_Ti -Tj
CommNet	10000	19061571	10001	0.78 ± 0.005	0.93 ± 0.049	0.78 ± 0.001	0.82 ± 0.002
PowerLaw	1000000	32280816	1001	2.07 ± 0.006	2.10 ± 0.012	2.06 ± 0.011	2.08 ± 0.06
Flickr-Days	2585570	33140018	135	1.31 ± 0.02	2.08 ± 0.31	1.31 ± 0.013	2.21 ± 0.2
Wiki-edits	21504191	266769613	134075025	0.40 ± 0.007	0.403 ± 0.001	0.39 ± 0.08	0.39 ± 0.008
Yahoo Netflow	32904819	1123508740	58735	2.19 ±0.45	1.51 ± 0.06	1.38 ±0.014	1.52 ±0.042

Graphs	Nodes	Contacts	Time Frames	neigh_TCBT_Ti	neigh_TCBT_Ti_Tj	Edge exists_CBT_Ti	Edge exists_CBT_Ti-Tj
CommNet	10000	19061571	10001	1.33 ± 1.44	1.43 ± 0.47	0.39 ± 0.66	0.39 ± 0.55
PowerLaw	1000000	32280816	1001	2.99 ± 0.55	5.70 ± 1.05	0.64 ± 0.12	1.6 ± 0.03
Flickr-Days	2585570	33140018	135	11.29 ± 8.52	38.49 ± 8.34	4.23 ± 2.81	5.44 ± 10.11
Wiki-edits	21504191	266769613	134075025	1.24, 1.911	1.42, 2.12	1.15, 0.18	1.15, 0.19
Yahoo Netflow	32904819	1123508740	58735	43.13, 0.263	51.21, 4.67	30.32, 2.36	31.2, 5.37

Graphs	Nodes	Contacts	Time Frames	neigh_CKD_Ti	neigh_CKD_Ti_Tj	Edge exists_CKD_Ti	Edge exists_CKD_Ti -Tj
CommNet	10000	19061571	10001	48.89 ± 11.56	64.46 ± 0.43	49.6 ± 3.4	49.7 ± 0.24
PowerLaw	1000000	32280816	1001	374.23 ± 50.72	374.64 ± 50.66	216.0 ± 5.3	226.13 ± 14.38
Flickr-Days	2585570	33140018	135	35.34 ± 10.39	45.22 ± 5.78	35.2 ± 1.2	37.2 ± 2.3
Wiki-edits	21504191	266769613	134075025	3.0 ± 3.0	4.39 ± 0.72	2.62 ± 1.7	2.98 ± 0.25
Yahoo Netflow	32904819	1123508740	58735	231.9 ± 82.1	254 ± 92.06	211.8 ± 89.0	212.32 ± 71

All the experiments were run on an Intel(R) Xeon(R) CPU E5520 @ 2.27GHz (16 Cores) with 64 GB of RAM, and the programs are written in GNU C/C++.

Conclusion

- Valuable insights can be gained from the analysis of time-evolving graphs.
- Our techniques show a significant reduction in memory requirement.
- All algorithms are tested on real-world datasets and show significant improvements over existing techniques.
- Our future work would focus on exploiting the parallelism in improving the compression techniques' timings in a broader domain of graphs..

Thank you.

Question?