GRAPH COMPRESSION

Prince Kumar(2020csb088)

Riya Manna (2020csb018)

Introduction

- Graphs are essential data structures used to represent complex relationships between entities in various domains.
- As graphs grow in size and complexity, there is a need to efficiently compress them for storage and transmission.
- Huffman coding is a popular technique for graph compression, where frequent symbols are represented with shorter codes, reducing the overall size of the graph representation.

Graph Compression using Huffman Coding

- Huffman coding is a lossless data compression algorithm that assigns variable-length codes to symbols based on their frequency of occurrence.
- In graph compression, nodes and edges are treated as symbols, and their frequencies are determined based on their occurrences in the graph.
- Nodes and edges with higher occurrences are assigned shorter codes, resulting in a more compact representation.
- The compressed graph can be efficiently reconstructed using the Huffman tree during decompression.

Benefits of Huffman Coding for Graph Compression

- Simple and efficient algorithm for lossless compression.
- Works well for graphs with skewed node/edge frequency distributions.
- Compression ratios are higher for graphs with long-tail distribution, such as power-law graphs.

Challenges of Huffman Coding for Graph Compression

- Building the Huffman tree introduces overhead during compression and decompression.
- Dynamic graphs with frequently changing structures pose a challenge as the Huffman tree needs to be rebuilt for each update.

Barabasi Graph

- The Barabasi graph is a type of random graph with a scale-free property.
- It is generated using preferential attachment, where new nodes attach to existing nodes with higher degrees.
- Barabasi graphs often exhibit power-law degree distributions, with a few nodes having a disproportionately high number of connections.

Power Law Graph

- Power law graphs have a degree distribution that follows a power-law, which means the probability of a node having degree k is proportional to k^(-α), where α is the power-law exponent.
- These graphs are characterized by a small number of high-degree nodes and a large number of low-degree nodes.

Graph Compression using Huffman Coding: Implementation and Analysis

Here is the code : Graph_Compression.cpp

The provided code is an implementation of graph compression using Huffman coding. The goal of graph compression is to efficiently represent a graph with fewer bits, reducing storage and transmission costs. The code takes an input graph, represented as an adjacency list, and calculates the frequencies of nodes and edges. It then applies Huffman coding to assign variable-length codes to nodes and edges based on their frequencies. The code calculates the total bits required for fixed-length coding and variable-length coding (with Huffman tree) to compare the compression efficiency. The output includes compressed ratios, encoding costs, efficiency, and statistical measures like minimum frequency, maximum frequency, mean, and standard deviation of node frequencies. The program demonstrates how Huffman coding can be used for graph compression and provides insights into the compression performance and statistical characteristics of the input graph.

Generating a Random Graph and Saving to a File

Here is the code : graphGenerator.py

The provided demonstrates the generation of a random graph using the G(n, p) model, where n is the number of nodes, and p is the density of edges. The networkx library is utilized for graph creation. Once the graph is generated, it is saved to a file in a specific format. The code starts by specifying the number of nodes and edges in the graph and then writes this information to a file named "input18.txt." Subsequently, the edges of the generated graph are written to the file in a human-readable format. This output file can be used for further analysis, graph compression experiments, or any other graph-related tasks. Additionally, the code includes a message to confirm that the file has been successfully written.

Generating a Barabasi-Albert Graph and Saving to a File

Here is the code : Barabasi_graph.py

The provided code demonstrates the generation of a Barabasi-Albert graph using the networkx library in Python. The Barabasi-Albert model is a preferential attachment model that creates graphs with a power-law degree distribution, which means a few nodes have a significantly higher number of connections than others. The code sets the number of nodes (n), the number of edges to attach from a new node to existing nodes (m), and a random seed for reproducibility.

Once the graph is generated, it is saved to a file named "input41.txt" in a specific format. The code first writes the number of nodes and edges in the graph to the file, followed by the edges of the generated graph in a human-readable format. This output file can be utilized for further analysis, graph compression experiments, or other graph-related tasks.

The code also includes commented-out sections that can be used to visualize the generated graph using matplotlib. However, this visualization is optional and can be disabled if the main purpose is to generate the graph and save it to a file. Finally, the code provides a message indicating that the file has been successfully written.

Generating a Power Law Cluster Graph and Saving to a File

Here is the code : Power_law_graph.py

The provided code showcases the generation of a power-law cluster graph using the networkx library in Python. The power-law cluster graph is a variation of the Watts-Strogatz model that combines the small-world property with a power-law degree distribution. This results in a graph with a few nodes having a significantly higher number of connections, forming clusters with long-tail degree distributions.

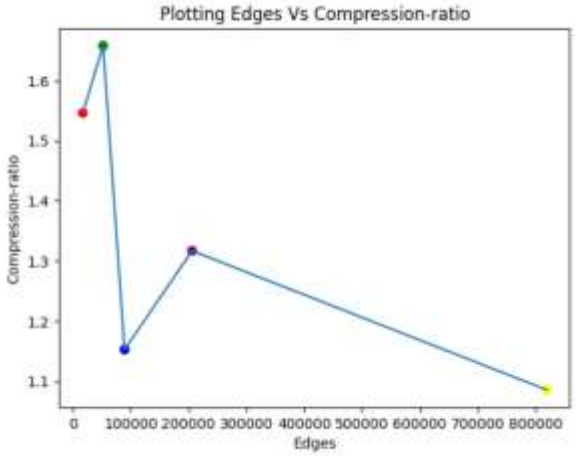
The code creates a power-law cluster graph with 1000 nodes, where each node is initially connected to its 200 nearest neighbors. The parameter 0.9 controls the probability of rewiring each edge to create the small-world effect.

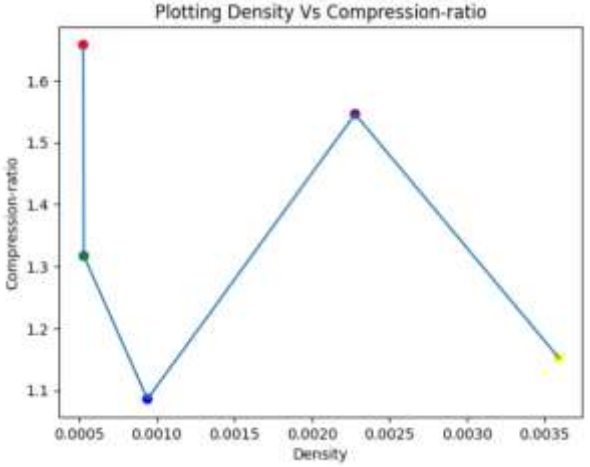
Once the graph is generated, it is saved to a file named "input39.txt" in a specific format. The code first writes the number of nodes and edges in the graph to the file, followed by the edges of the generated graph in a human-readable format. The output file can be used for further analysis, graph compression experiments, or other graph-related tasks.

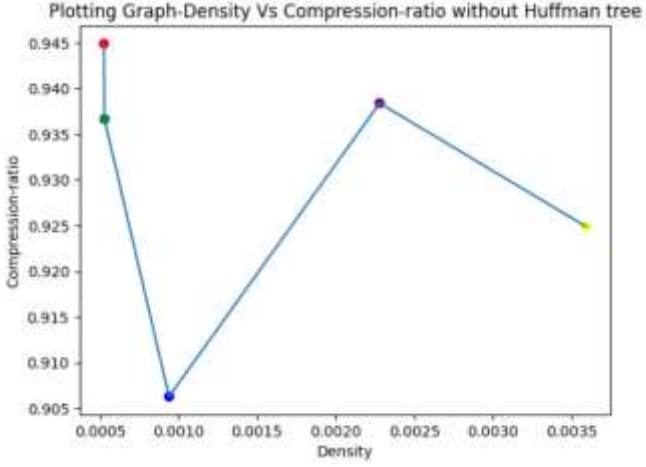
The code also includes commented-out sections that can be used to visualize the generated graph using matplotlib. However, this visualization is optional and can be disabled if the main purpose is to generate the graph and save it to a file. Finally, the code provides a message indicating that the file has been successfully written.

Name of Graph data set	Number of Nodes	Number of Edges	Density	Fixed Length of Encoding cost	Encoding cost of compressed Graph	Encoding cost of Huffman tree	Compressed Ratio
Artist	50,515	8,19,306	0.000939974	27026032	29343067	4849360	1.08573
Company	14,113	52,310	0.000525299	1662262	2756044	1185422	1.65801
Government	7,057	89,455	0.003592 99	2417571	2786379	550381	1.15255
New Sites	27,917	2,06,259	0.000529323	6606525	8700293	2512455	1.31692
TV Shows	3,892	17,262	0.002279 75	460992	712745	280164	1.54611

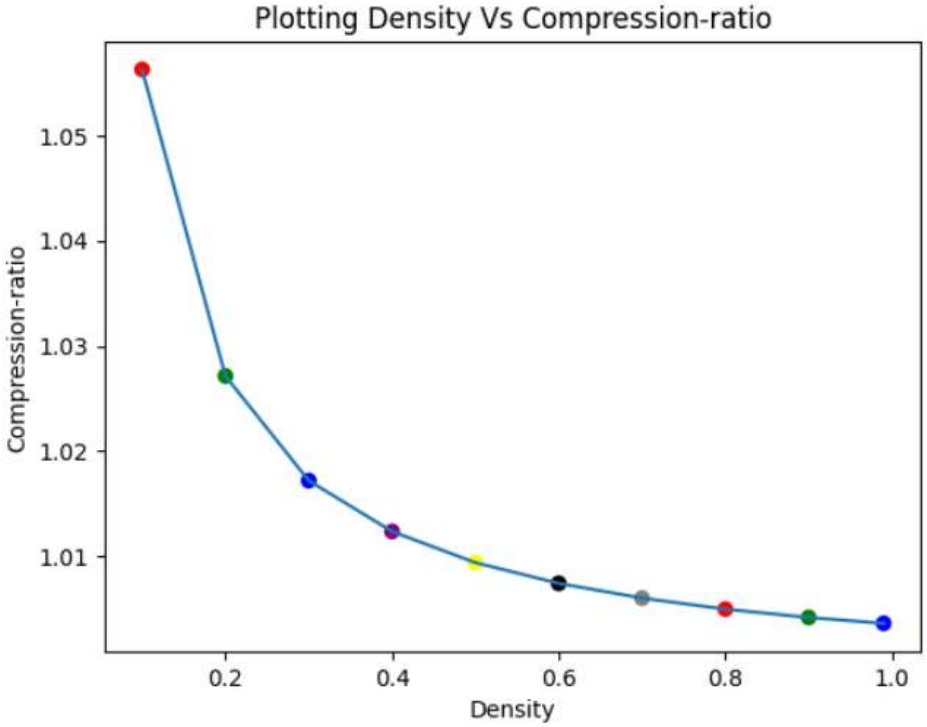
Name of Graph data set	Number of Nodes	Number of Edges	Density	Fixed Length of Encoding cost	Encoding cost of compressed Graph	Encoding cost of Huffman tree	Compressed Ratio	Total bits in Huffman tree coding / total bits in variable length coding	compression ratio without huffman tree
output1	100	4683	0.95	66262	67703	4165	1.02175	6.15	0.95889
output2	100	4897	0.99	69258	70623	4165	1.01971	5.89	0.959571
output3	500	118509	0.95	2137662	2158796	26955	1.00989	1.24	0.997277
output4	500	123502	0.99	2227536	2248497	26955	1.00941	1.19	0.997309
output5	1000	474547	0.95	9500940	9537698	59950	1.00387	0.62	0.997559
output6	1000	494326	0.99	9896520	9932553	59950	1.00364	0.6	0.997583
output7	2000	1979017	0.99	43560374	43596803	131945	1.00084	0.3	0.997807
output8	3000	4274198	0.95	102616752	99695304	215940	0.97153	0.21	0.969426


Plotting Edges Vs Compression-ratio
Compression-ratio
1.6
1.5
1.4
1.3
1.2
1.1
0
100000
200000
300000
400000
500000
600000
700000
800000
Edges


Plotting Density Vs Compression-ratio
Compression-ratio
1.6
1.5
1.4
1.3
1.2
1.1
0.0005
0.0010
0.0015
0.0020
0.0025
0.0030
0.0035
Density

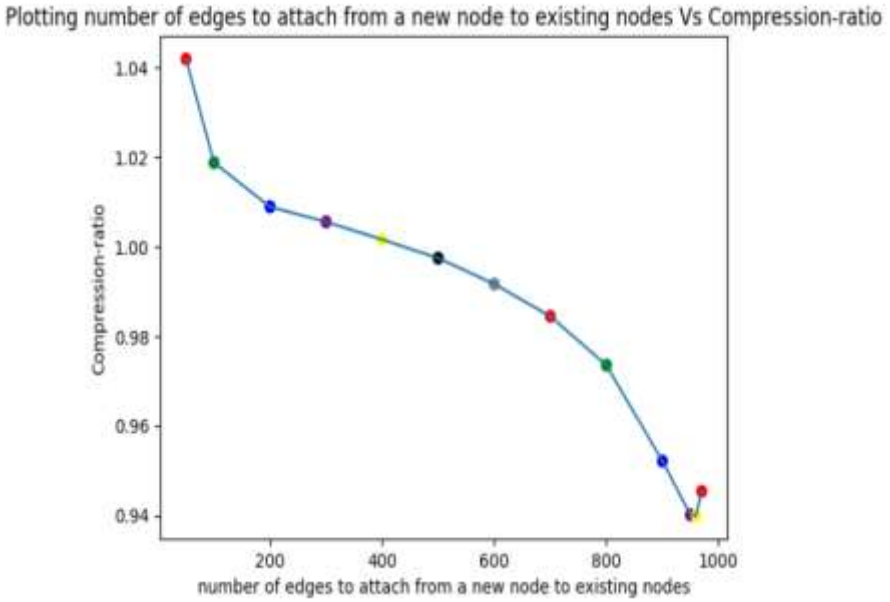

Plotting Graph-Density Vs Compression-ratio without Huffman tree
Compression-ratio
0.945
0.940
0.935
0.930
0.925
0.920
0.915
0.910
0.905
0.0005
0.0010
0.0015
0.0020
0.0025
0.0030
0.0035
Density

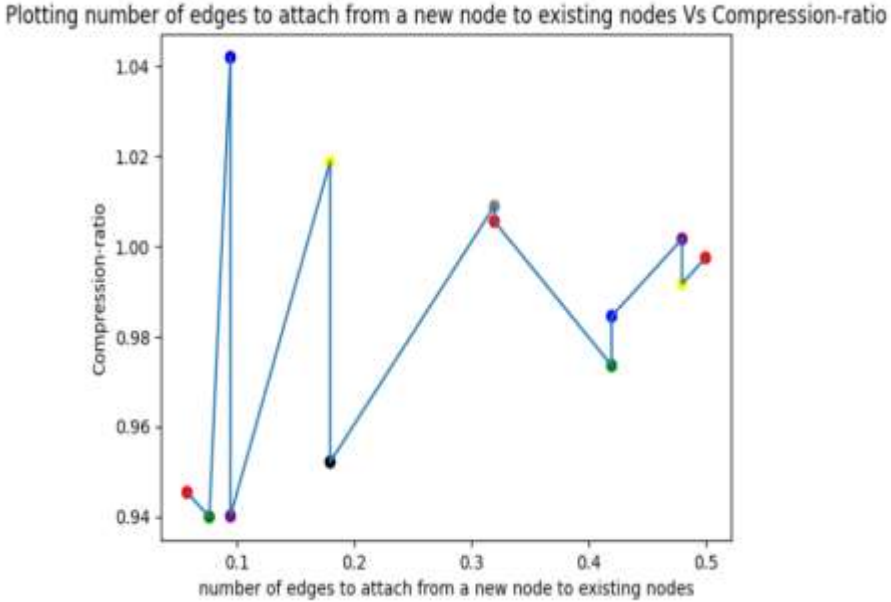
Name of Graph data set	Number of Nodes	Number of Edges	Density	Fixed Length of Encoding cost	Encoding cost of compressed Graph	Encoding cost of Huffman tree	Compressed Ratio	Total bits in Huffman tree coding / total bits in variable length coding	compression ratio without huffman tree
output9	1000	50132	0.1	1012640	1069608	59950	1.05626	5.6	0.997
output10	1000	99802	0.2	2006040	2060482	59950	1.02714	2.9	0.997254
output11	1000	150265	0.3	3015300	3067157	59950	1.0172	1.95	0.997316
output12	1000	199724	0.4	4004480	4054029	59950	1.01237	1.47	0.997403


Plotting Density Vs Compression-ratio
Compression-ratio
1.05
1.04
1.03
1.02
1.01
0.2
0.4
0.6
0.8
1.0
Density

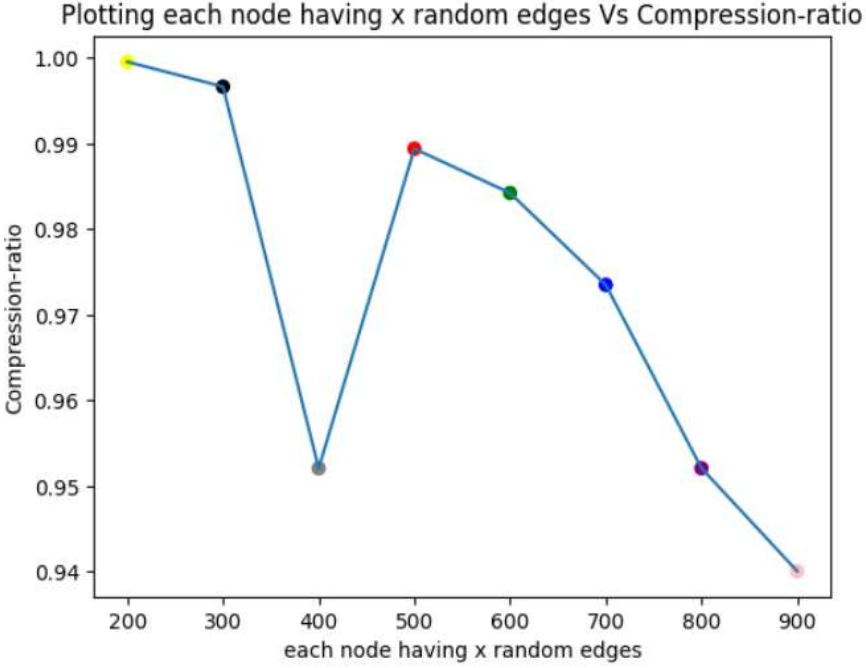
ADD TITLE:

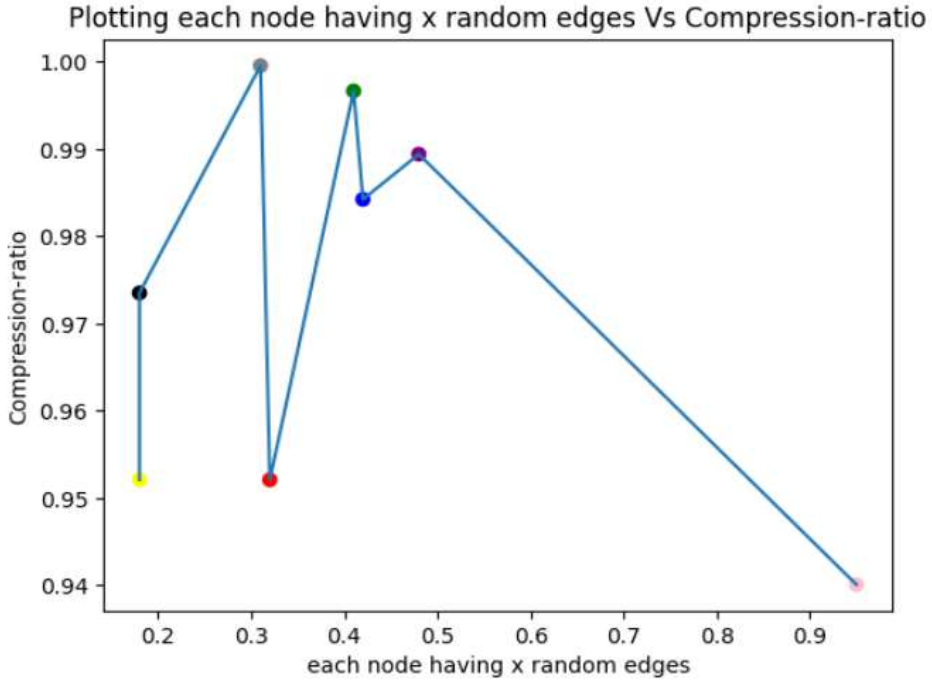
Name of Graph data set	Number of Nodes	Number of Edges	Density	Fixed Length of Encoding cost	Encoding cost of compressed Graph	Encoding cost of Huffman tree	Compressed Ratio	Total bits in Huffman tree coding / total bits in variable length coding	compression ratio without huffman tree	m	Minimum frequency	Maximum Frequency	standard Deviation	Mean
output19	1000	47500	0.095	960000	1000257	59950	1.04193	5.99	0.979486	50	50	381	57.5658	95
output20	1000	90000	0.18	1810000	1844009	59950	1.01879	3.25	0.985668	100	146	706	123.515	320
output21	1000	160000	0.32	3210000	3238554	59950	1.0089	1.85	0.990219	200	235	830	153.144	420
output22	1000	210000	0.42	4210000	4233308	59950	1.00554	1.41	0.991296	300	235	830	153.144	420
output23	1000	240000	0.48	4810000	4817645	59950	1.00159	1.24	0.989126	400	247	922	187.357	480
output24	1000	250000	0.5	5010000	4997116	59950	0.997428	1.199	0.985462	500	238	965	225.126	500
output25	1000	240000	0.48	4810000	4769913	59950	0.991666	1.25	0.979202	600	239	989	259.917	480
output26	1000	210000	0.42	4210000	4144584	59950	0.984462	1.44	0.970222	700	206	998	284.402	420
output27	1000	160000	0.32	3210000	3124973	59950	0.973512	1.91	0.954836	800	154	999	291.076	320
output28	1000	90000	0.18	1810000	1723300	59950	0.952099	3.478	0.918978	900	85	999	256.802	180
output29	1000	47500	0.095	960000	902493	59950	0.940097	6.64	0.877649	950	44	999	201.901	95
output30	1000	38400	0.077	778000	731248	59950	0.939907	8.198	0.862851	960	36	999	184.372	76.8
output31	1000	29100	0.058	592000	559646	59950	0.945348	10.71	0.844081	970	26	999	162.992	58.2


Plotting number of edges to attach from a new node to existing nodes Vs Compression-ratio
Compression-ratio
1.04
1.02
1.00
0.98
0.96
0.94
200
400
600
800
1000
number of edges to attach from a new node to existing nodes


Plotting number of edges to attach from a new node to existing nodes Vs Compression-ratio
Compression-ratio
1.04
1.02
1.00
0.98
0.96
0.94
0.1
0.2
0.3
0.4
0.5
number of edges to attach from a new node to existing nodes

Name of Graph data set	Number of Nodes	Number of Edges	Density	Fixed Length of Encoding cost	Encoding cost of compressed Graph	Encoding cost of Huffman tree	Compressed Ratio	Total bits in Huffman tree coding / total bits in variable length coding	compression ratio without huffman tree	m
output32	1000	90000	0.18	1810000	1723299	59950	0.952099	3.47	0.918977	400
output33	1000	240000	0.48	4810000	4759045	59950	0.989406	1.25	0.976943	500
output34	1000	2100000	0.42	4210000	4143609	59950	0.98423	1.44	0.96999	600
output35	1000	160000	0.32	3210000	3124966	59950	0.97351	1.91	0.954834	700
output36	1000	90000	0.18	181000	1723303	59950	0.952101	3.47	0.91898	800
output37	1000	47500	0.95	960000	902493	59950	0.940097	6.64	0.877649	900
output38	1000	206867	0.41	4147340	4133435	59950	0.996647	1.45	0.982192	300


Plotting each node having x random edges Vs Compression-ratio
Compression-ratio
1.00
0.99
0.98
0.97
0.96
0.95
0.94
200
300
400
500
600
700
800
900
each node having x random edges


Plotting each node having x random edges Vs Compression-ratio
Compression-ratio
1.00
0.99
0.98
0.97
0.96
0.95
0.94
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
each node having x random edges

Conclusion

- Graph compression is a crucial area of research for managing large and evolving graphs efficiently.
- Huffman coding and other compression techniques play a significant role in reducing storage and transmission costs.
- As the demand for processing graphs grows, further advancements in compression algorithms are expected.

THANK YOU