Assignment 2 Report

FOR

CS F364 DESIGN AND ANALYSIS OF ALGORITHM

BY

Names of the Student	ID No.
• DHAIRYA LUTHRA	2022A7PS1377H
• ANIMISH AGRAHARI	2022A7PS1367H
• SHREEJEET MISHRA	2022A7PS0036H
• SAUMYA GAUR	2022A7PS1318H
• AARIV WALIA	2022A7PS0052H

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)

HYDERABAD CAMPUS

(APRIL 2025)

Acknowledgement

We would like to thank BITS Pilani Hyderabad Campus and Prof. Apurba Das for their invaluable guidance and support throughout the assignment and our learning process. Their lectures have enhanced our understanding. This assignment helped in strengthening our analytical skills and knowledge.

1.INTRODUCTION

1.1 What is Clique-Density Subgraph (CDS) ?

Given a graph G (V , E) G(V,E) and a fixed h h-clique Ψ Ψ, the CDS is a subgraph D (V D , E D) D(V D ,E D) of G G that maximizes the h h-clique-density with respect to Ψ Ψ2. The h h-clique-density is the number of h h-cliques in the subgraph divided by the number of vertices in the subgraph, i.e., it measures how densely h h-cliques are packed within the subgraph.

Key points from the notation table:

- D(V D ,E D): The CDS whose h h-clique-density is ρ ρ
- ρ (D , Ψ) ρ(D,Ψ): The h h-clique-density of subgraph D D with respect to Ψ Ψ

A CDS is the subgraph of G G that contains the highest possible density of h h-cliques, making it a powerful tool for identifying tightly interconnected communities or patterns within a network

1.2 What is Minimum s-t Cut: ?

The smallest set of edges whose removal disconnects the source **s.** from the sink **t,** in the flow network. The value of this cut equals the maximum possible flow from **s** to **t.**

1.3 What is Flow Network :

A directed graph used to model the problem, including a source node **s**, a sink node **t**, and possibly intermediate nodes. Each edge has a capacity, and the flow through an edge cannot exceed its capacity

1.4 Problem Statement:

The algorithm is designed to solve the Densest Subgraph Discovery (DSD) problem with respect to edge- and clique-density. Specifically, it aims to find, in a given graph G (V , E) G(V,E), the subgraph D that has the highest density according to one of two measures:
Edge-Density Subgraph (EDS): The subgraph with the highest average number of edges per vertex. Clique-Density Subgraph (CDS): The subgraph with the highest average number of h h-cliques (complete subgraphs of h h vertices) per vertex.

2. EXPLORING RESEARCH PAPERS

2.1 Clique Densest Subgraph (CDS) Algorithm Description

This algorithm solves the Clique Densest Subgraph problem, which aims to find a dense subgraph of a given graph G = (V, E).

Key Concepts:

The algorithm constructs a flow network and applies a minimum cut algorithm to identify dense subgraphs. It takes as input:

- An undirected graph G = (V, E)
- A density parameter ψ
- Edge weights E_1 and E_2

Algorithm Outline:

1. **Initialization Phase:**
    - Sets a threshold parameter t based on the maximum degree of vertices in V multiplied by ψ
    - Initializes Λ with all (h-1)-cliques in G
    - Sets D to an empty set (to track dense subgraphs)
2. **Flow Network Construction:**
    - Creates a network with source s and sink t
    - Adds vertices from V with appropriate connections
    - Adds edges between (h-1)-cliques and vertices
    - Sets capacity values based on degree and ψ parameters
3. **Minimum Cut Computation:**
    - For each (h-1)-clique v, finds the minimum s-t cut in the constructed flow network
    - If the minimum cut value is finite, adds the corresponding subset S(v) to the solution set
4. **Result:**
    - Returns D, the collection of dense subgraphs identified through the flow network approach

5. **Efficiency:**

This algorithm is particularly effective for identifying dense subgraph structures through network flow techniques. The flow network construction enables the identification of subgraphs that maximize density according to the parameter ψ.

The exact complexity depends on the implementation of the minimum cut algorithm used and the number of (h-1)-cliques in the graph.

2.2 CoreExact Algorithm Description

This algorithm presents an optimized approach to solving the Clique Densest Subgraph (CDS) problem by incorporating core decomposition techniques. It takes the same input as the previous algorithm but employs a more sophisticated strategy.

Key Concepts:

The CoreExact algorithm combines core decomposition with flow network techniques to efficiently identify dense subgraphs. It leverages the structure of the graph to reduce computational overhead by focusing on promising regions.

Algorithm Outline:

1. **Core Decomposition:**
    - Performs initial core decomposition using Algorithm 3 (not shown in the image)
    - Locates the (k*, ψ)-core using pruning techniques
    - Initializes set C ← ∅, D ← ∅, U ← ∅, and t ← q*, where q* = ψ·kmax
2. **Component Analysis:**
    - For all connected components of the (k*, ψ)-core, performs further analysis
    - If component size is larger than k*, adds the component to set C
    - For each component in C:
        - Builds a flow network F(VF, EF) following the same approach as Algorithm 1
        - Finds minimum s-t cut from this flow network
3. **Iterative Refinement:**
    - While t ≥ 1/(|C|·(|C|-1)), continues refinement:
        - Updates t parameter based on current values
        - Rebuilds the flow network and finds minimum cuts
        - If cut value is greater than |T|, removes vertices according to S(v)
        - Otherwise, adds the induced subgraph G[U] to the solution set D
4. **Result:**
    - Returns D, the collection of identified dense subgraphs

5. **Efficiency:**

This algorithm improves upon the basic approach by:

- Using core decomposition to identify promising regions first
- Processing components separately, reducing the complexity of flow network operations

- Employing iterative refinement based on minimum cut results

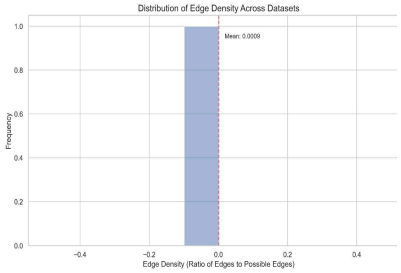
3. IMPLEMENTATION AND OBSERVATION

We have implemented all 2 of the above algorithms and have included the code files on our github repository : https://github.com/Kal-El-pt2/DAA_Assignment_part_2_final

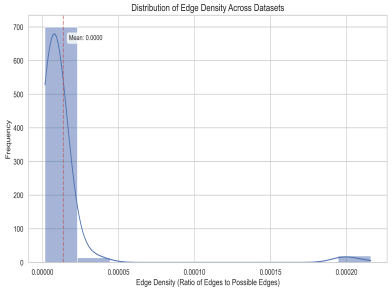
Here are the results :

3.1 Edge-Density Graph For Algorithm 1 for h=2

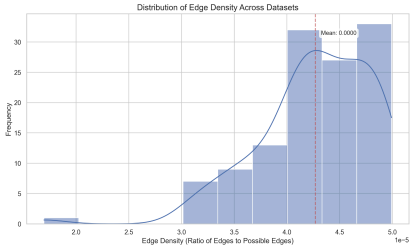
Dataset1 :
Yeast
Range: .0009 to 0.0017
Mean:0.0012
Peaks:0.0011
Minimum:0.0009



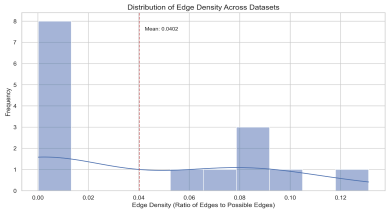
Dataset2:
As-733
Range:2.1*10^-4
Mean:1.5 *10^-5
Peaks:1.2*10^-5
Minimum:0.0



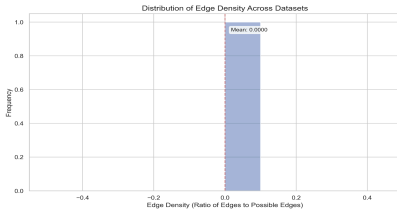
Dataset3 :
As-Caida
Range:3.0*10^-5
Mean:4.2 *10^-5
Peaks: 4.3*10^-5
Minimum:1.9*10^-5



Dataset4 :
Netscience
Range:0.125
Mean:0.0402
Peaks:0.005
Minimum:0.0



Datset5:
Ca-HepTh
Range:0
Mean:0.0
Peak:0.0
Minimum:0.0



3.2 execution_time_vs_graph_size for Algorithm 1 for h=2

Dataset1 :

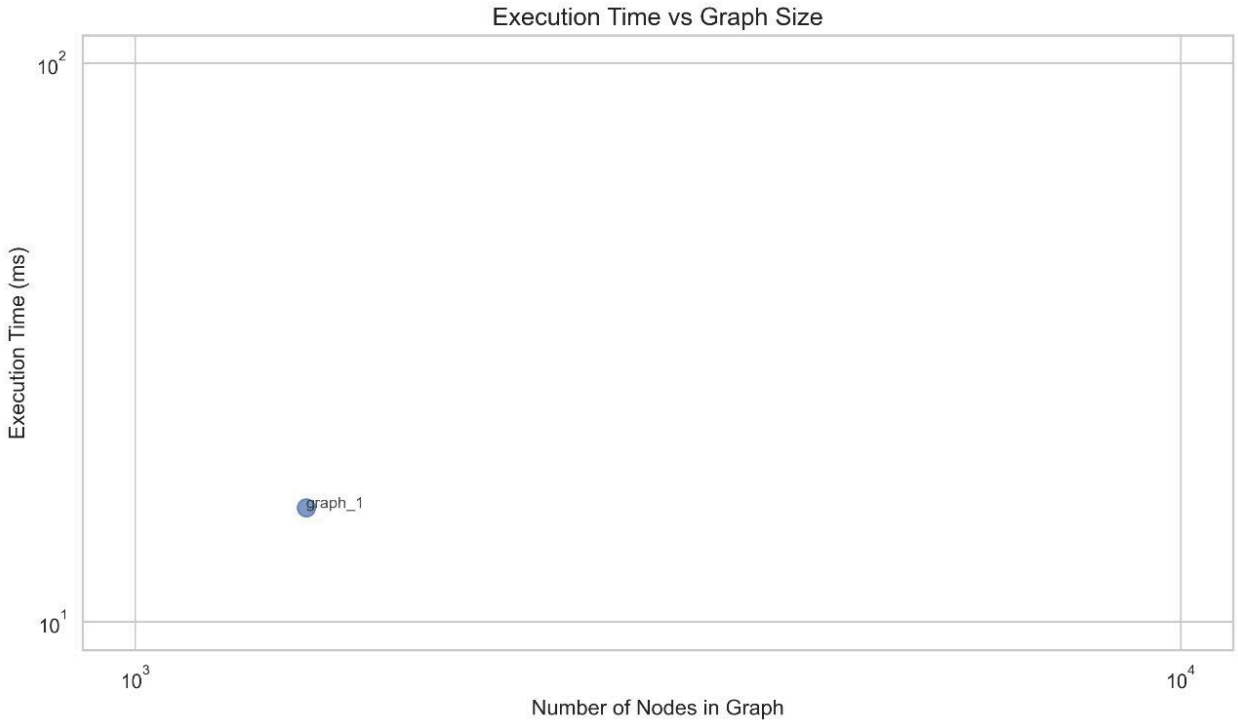
Yeast

Slope: 1.84

Node Range: 2.0*10^4 to 3.0*10^4

Execution Time Range:1800ms to 6000ms

Outliers:dip201001010.mif25



Dataset2 :

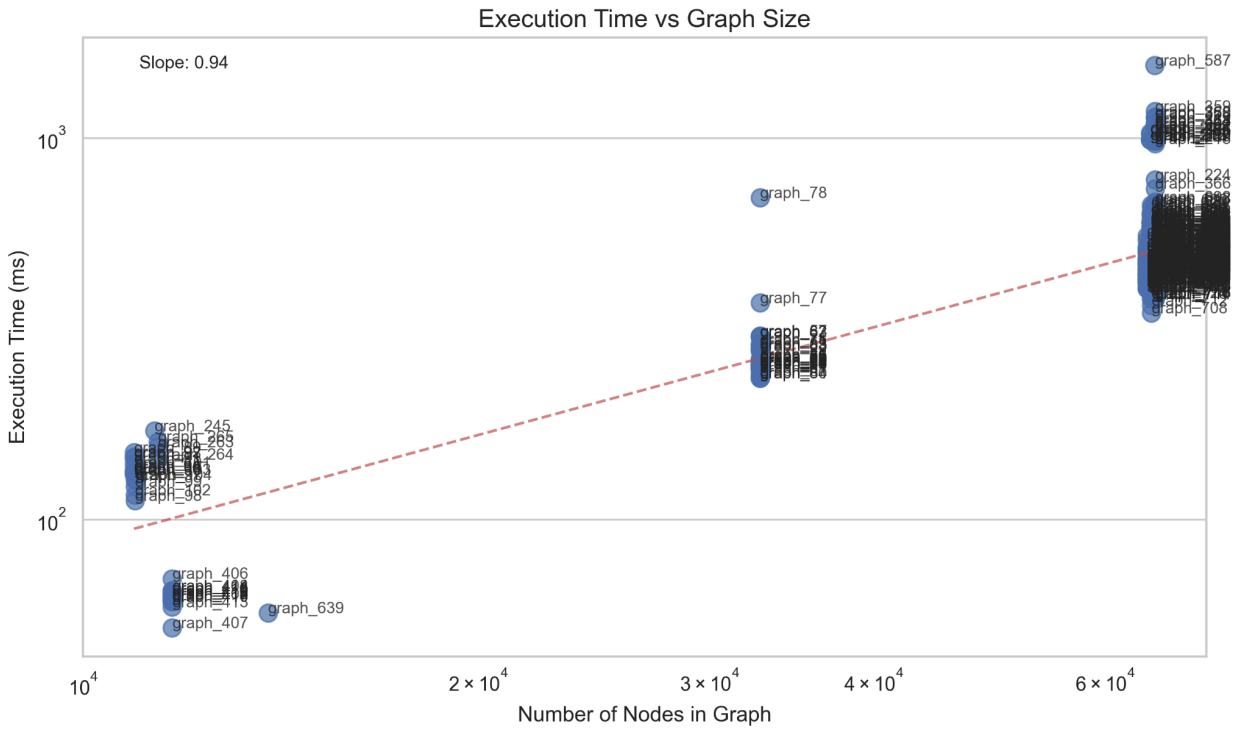
As-733

Slope: 0.94

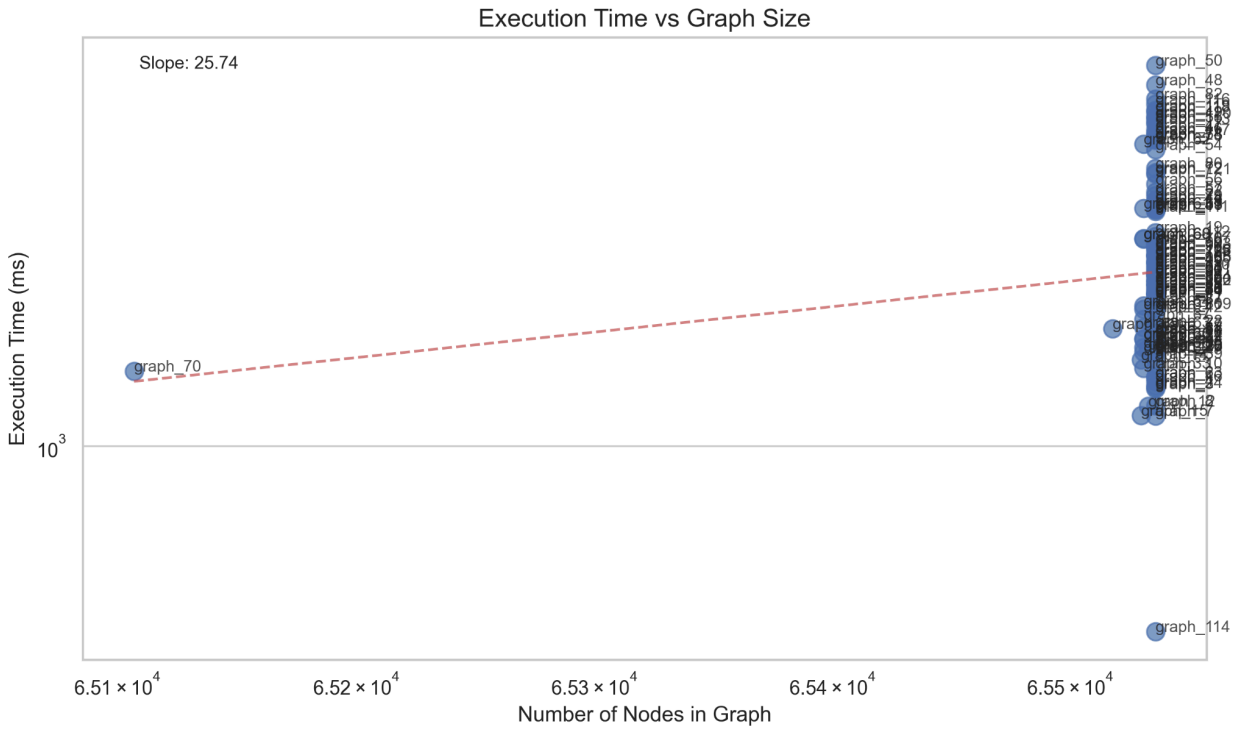
Node Range: 1.0*10^4 to 6.0 *10^4

Execution Time Range:50 ms to 1500ms

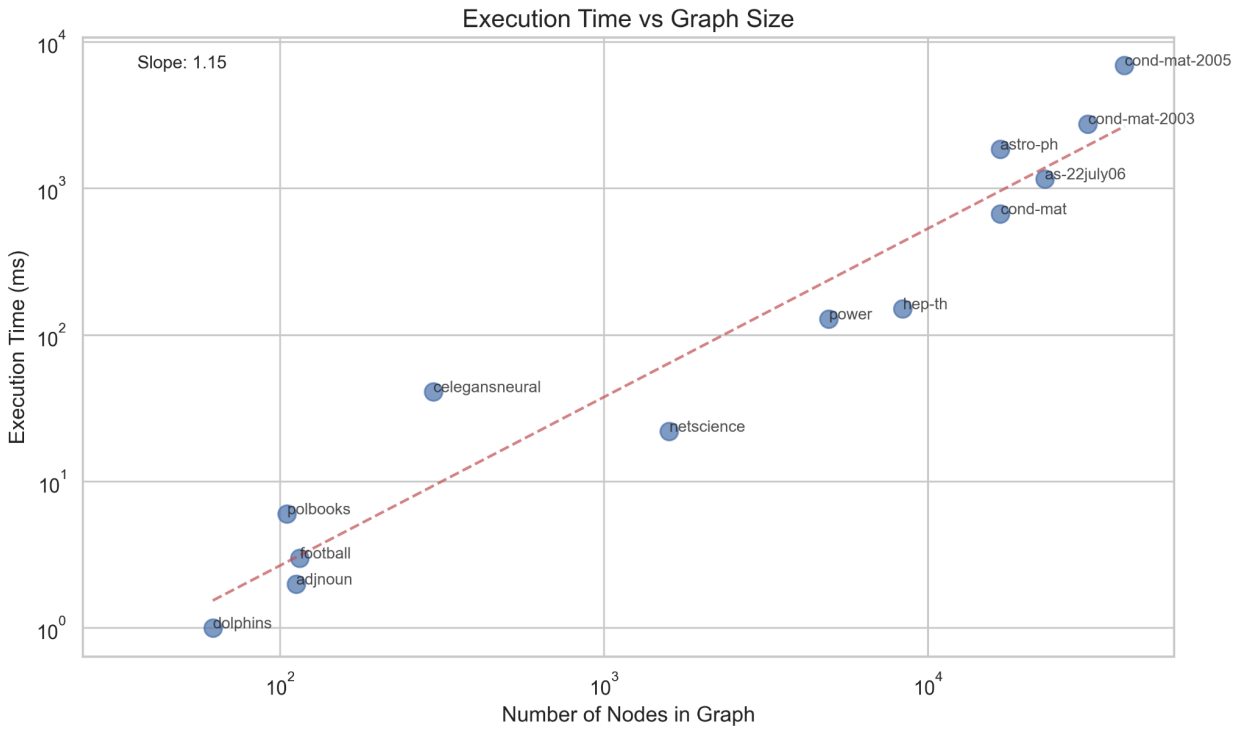
Outliers:Graph_78,graph_587



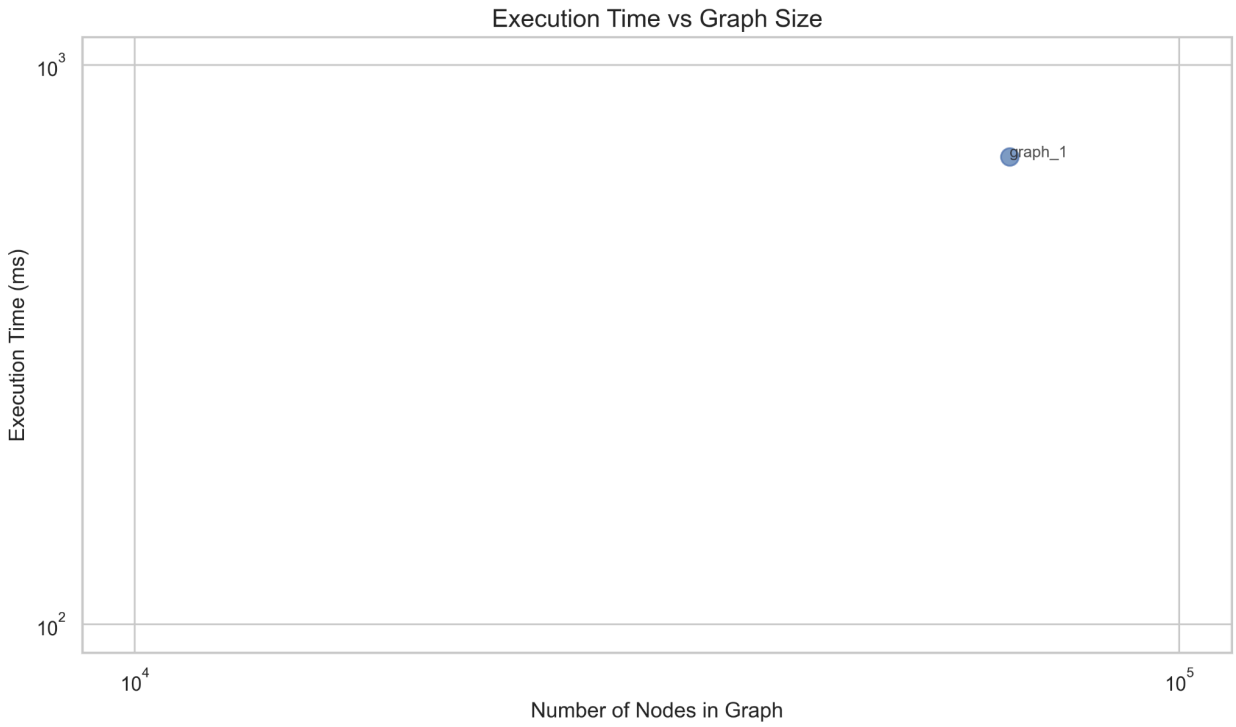
Dataset3 :
As-Caida
Slope: 25.74
Node Range: 6.51*10^4 to 6.55 * 10^4
Execution Time Range:1000ms to 5000ms
Outliers:Graph_70,graph_114



Dataset4 :
Netscience
Slope: 1.15
Node Range: 10^2 to 10^4
Execution Time Range:1ms to 8000ms
Outliers:netscience,celegansneural

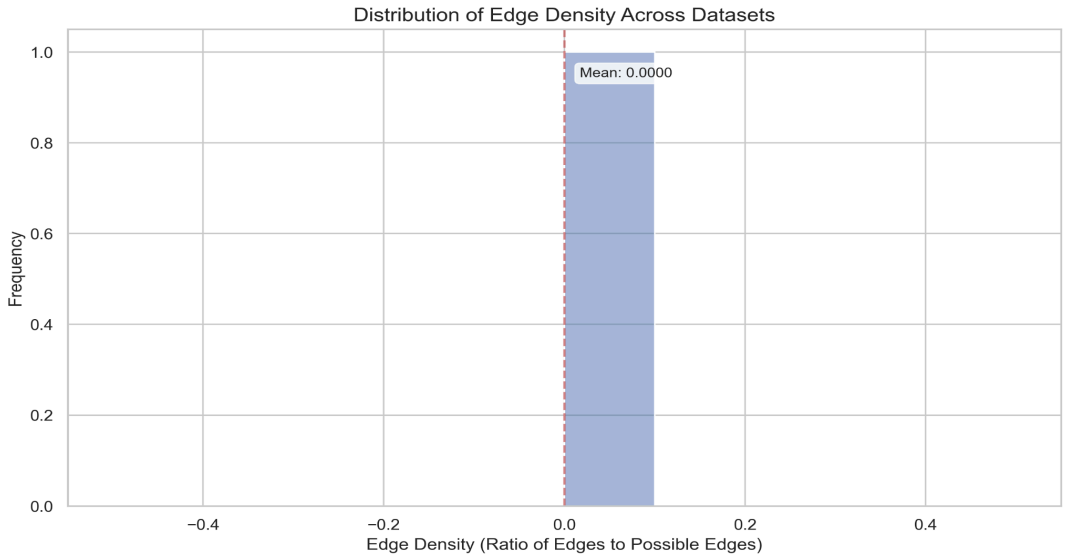


Dataset5 :
Ca-HepTh
Slope: not possible only one data
Node Count: 5.5*10^4
Execution Time Range:500ms
Outliers:none

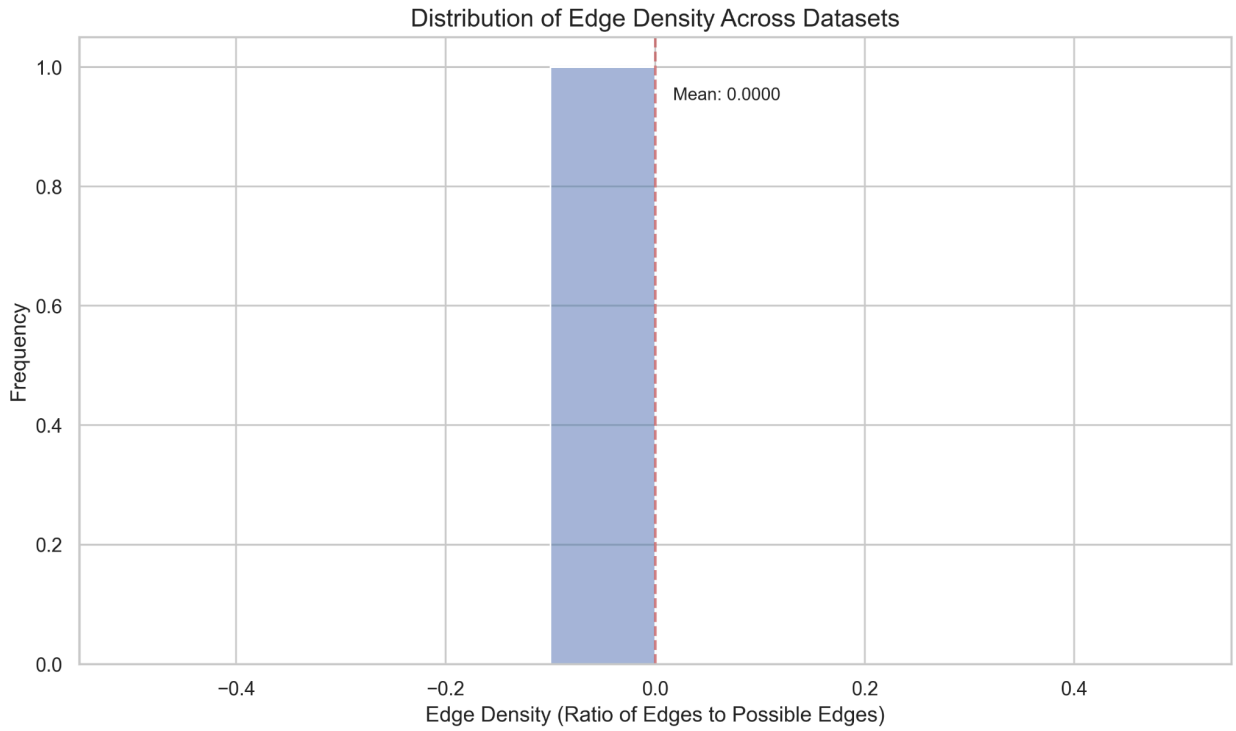


3.3 Edge-Density Graph For Algorithm 4

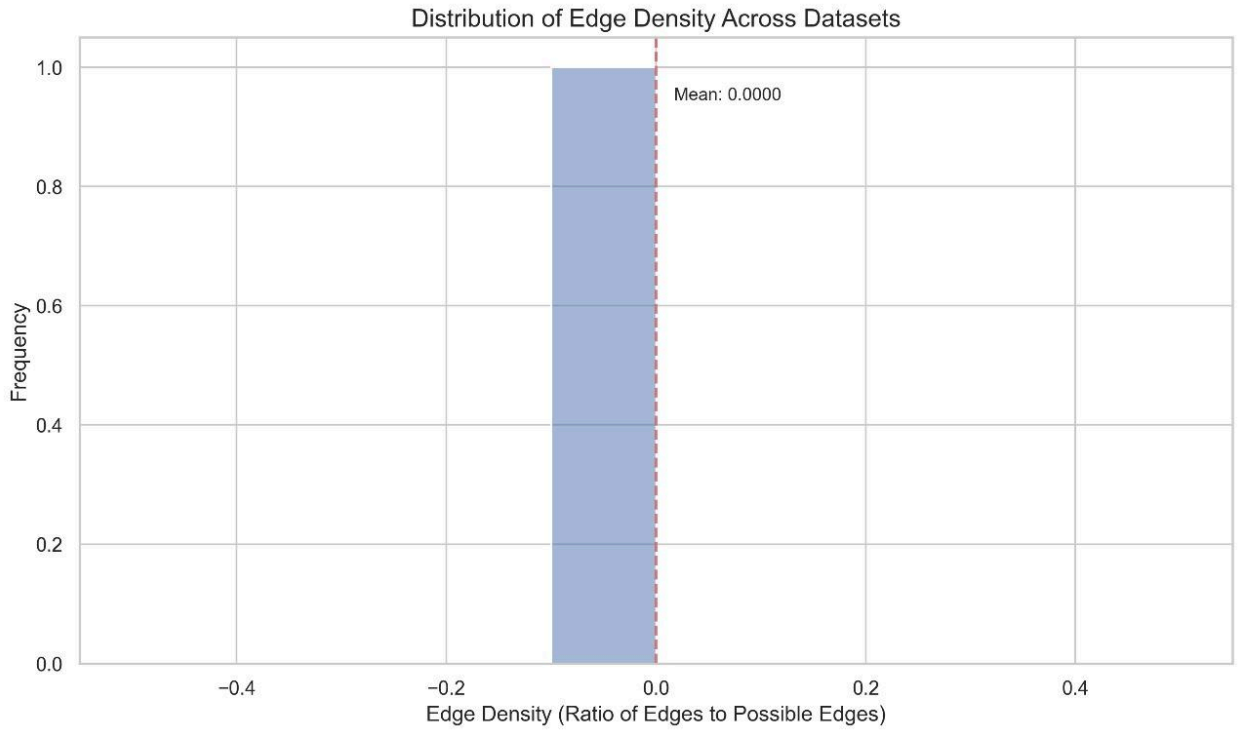
Dataset1 :
Ca-HepTh
h=2



h=3


Distribution of Edge Density Across Datasets
Mean: 0.0000
Frequency
1.0
0.8
0.6
0.4
0.2
0.0
−0.4
−0.2
0.0
0.2
0.4
Edge Density (Ratio of Edges to Possible Edges)

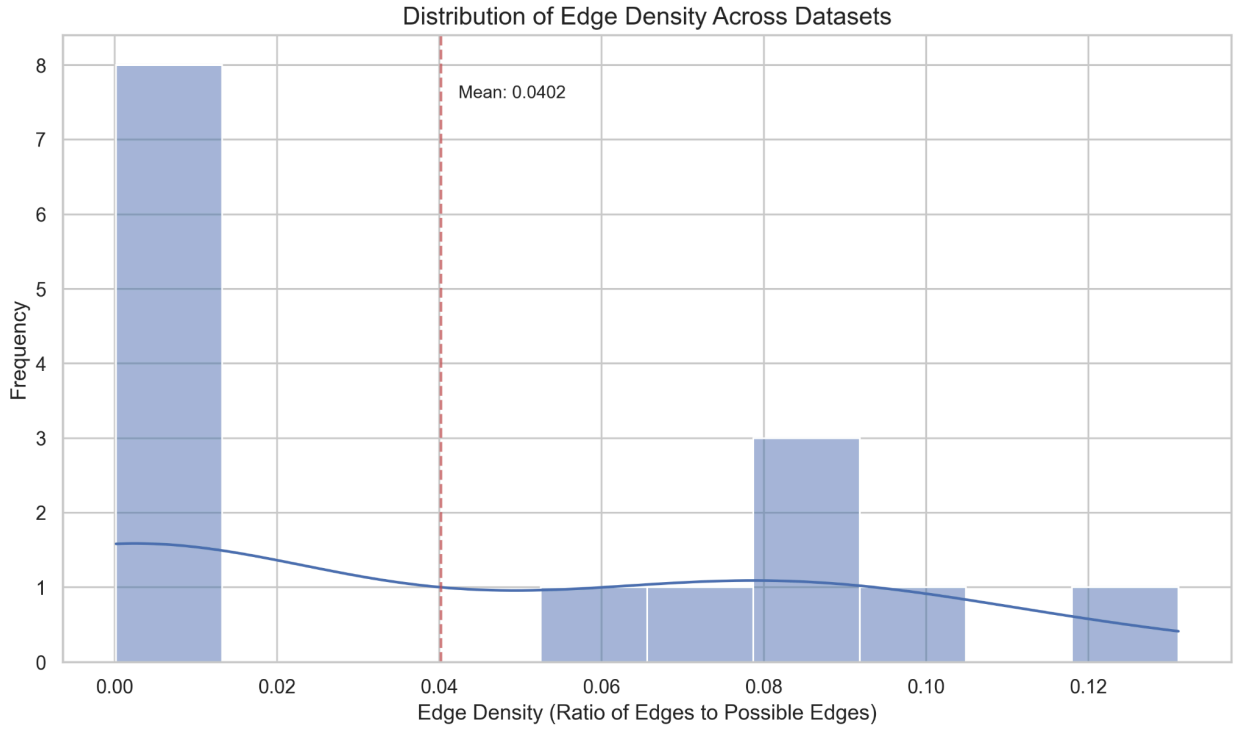
h=4



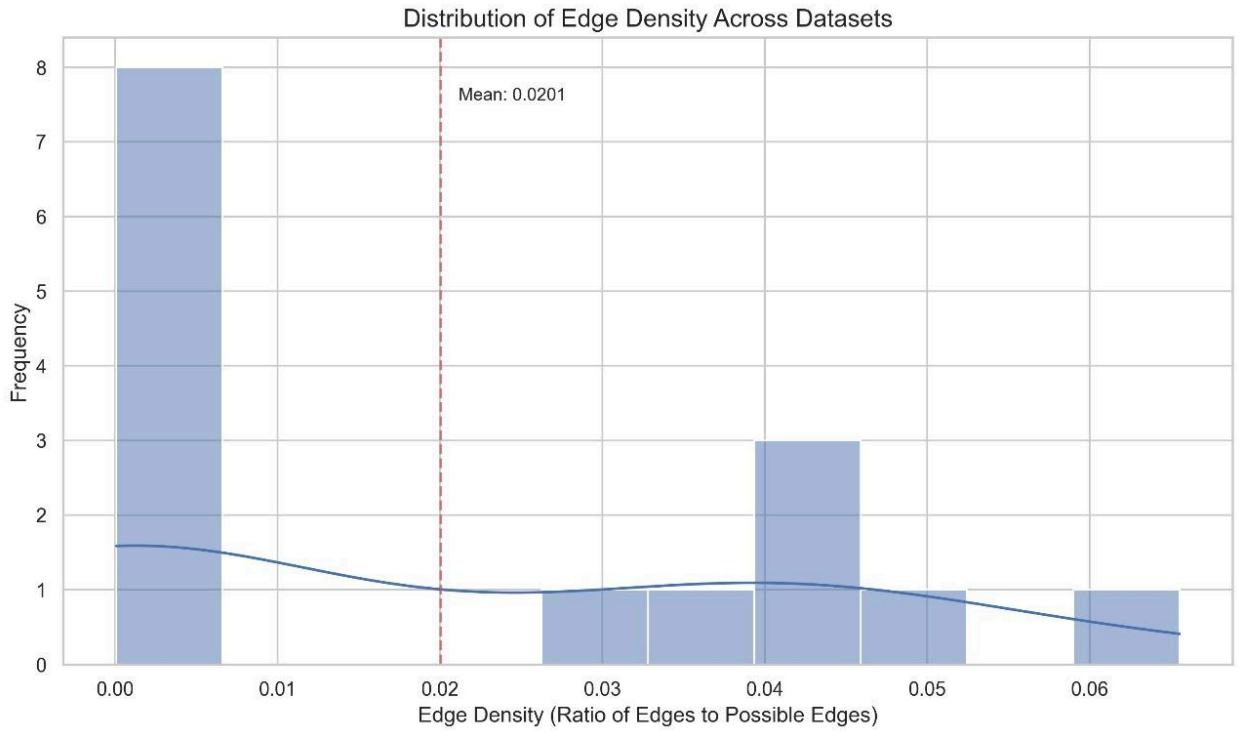
Dataset2 :

Netscience

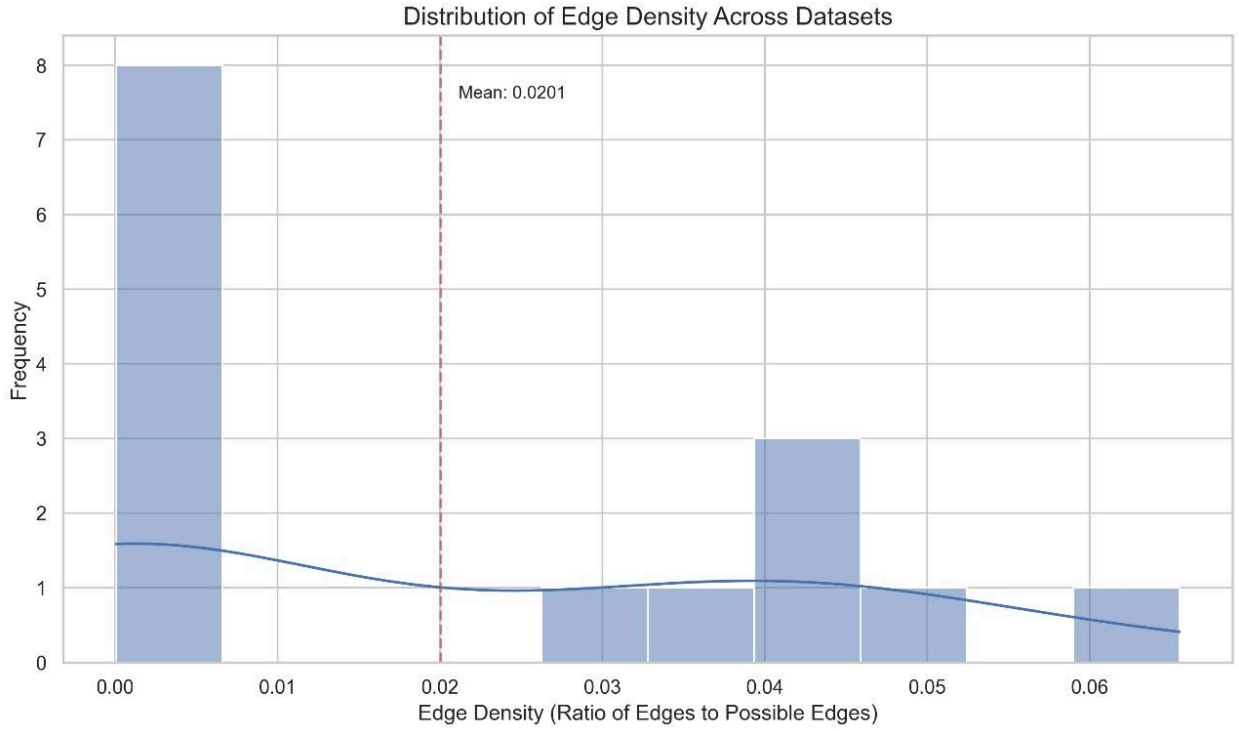
h=2



h=3



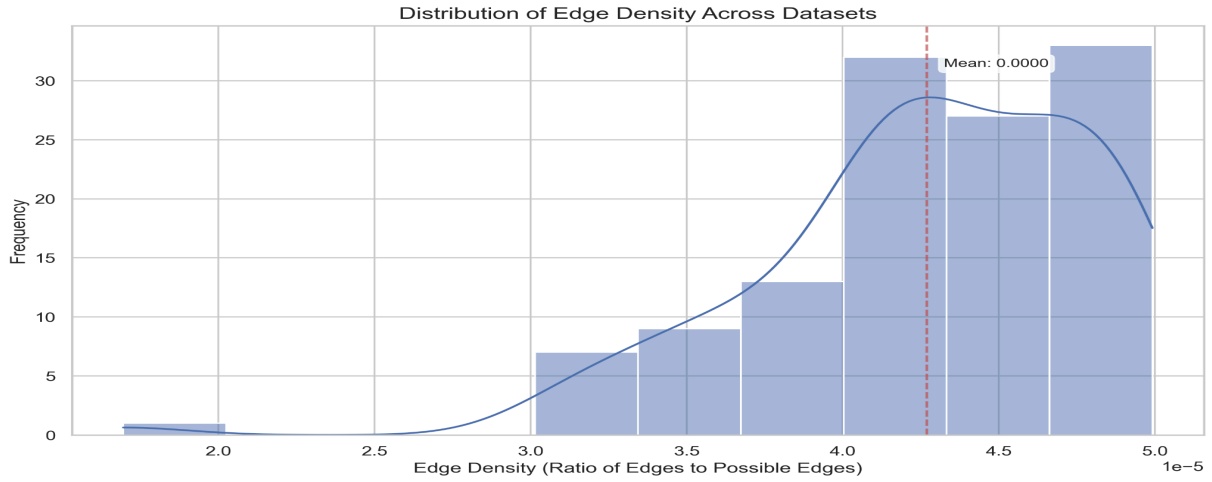
h=4



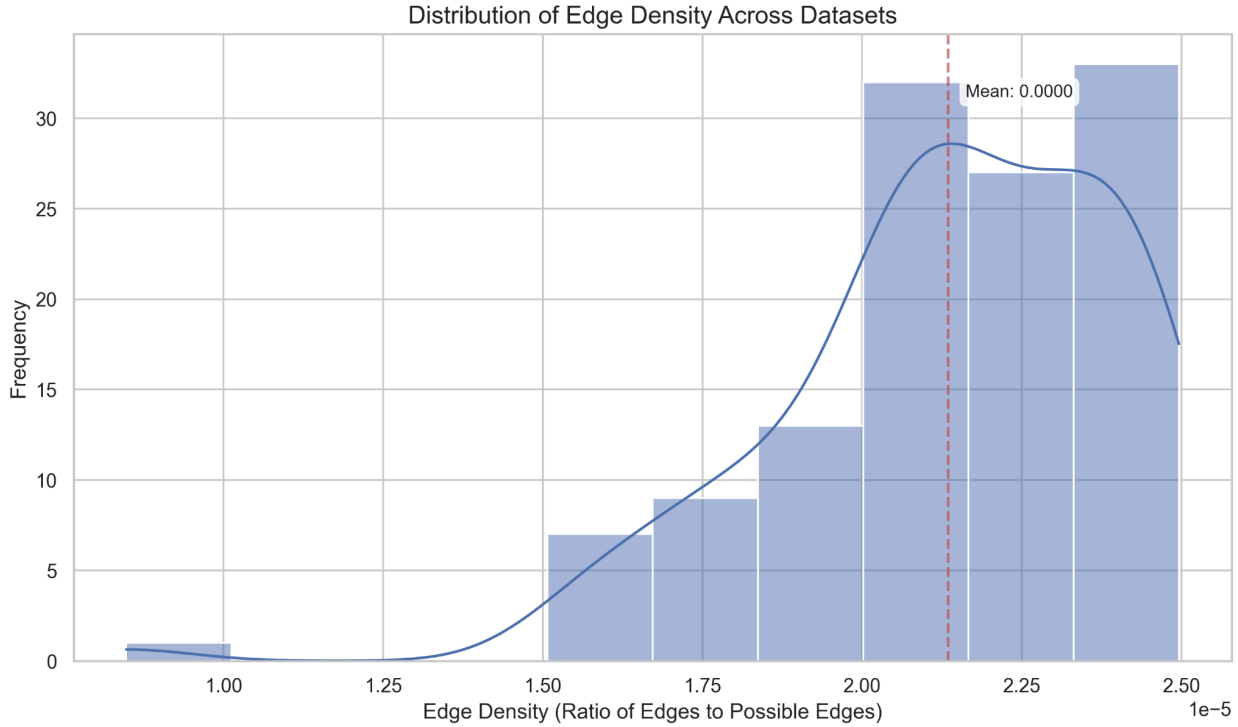
Dataset3 :

As-Caida

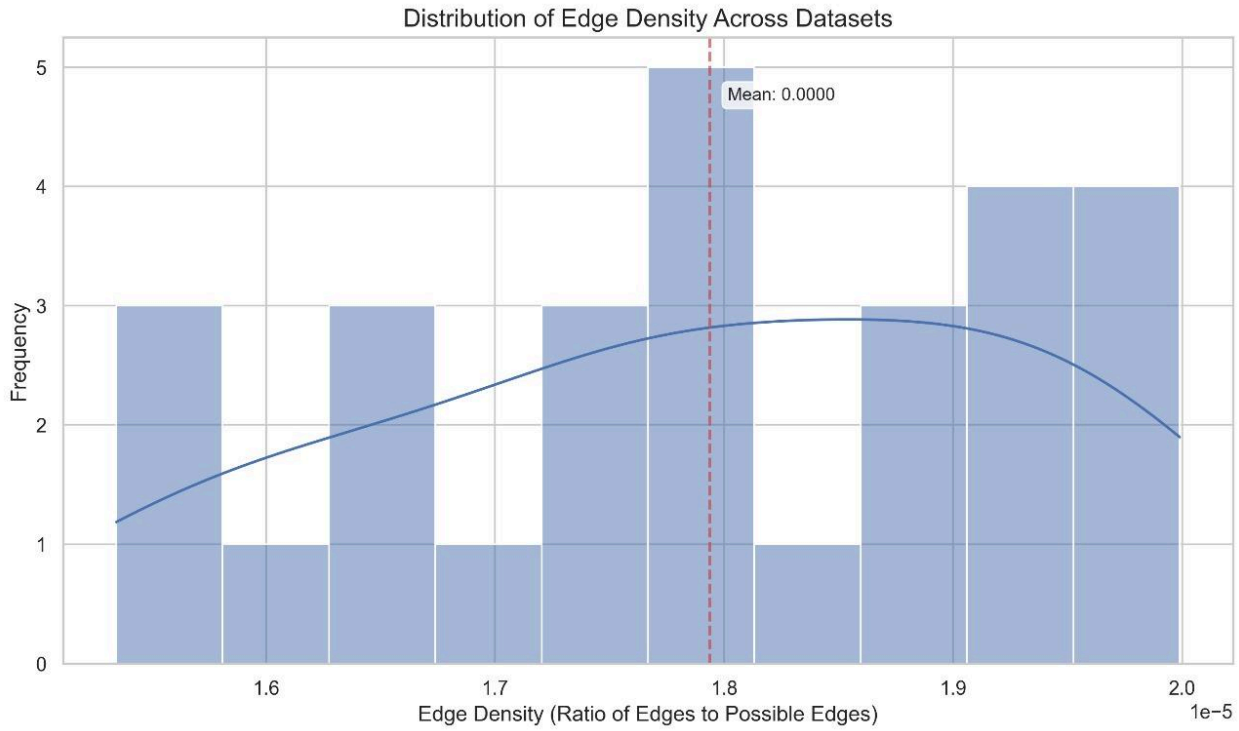
h=2


Distribution of Edge Density Across Datasets
Mean: 0.0000
Frequency
0
5
10
15
20
25
30
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Edge Density (Ratio of Edges to Possible Edges)
1e−5

h=3


Distribution of Edge Density Across Datasets
Mean: 0.0000
Frequency
0
5
10
15
20
25
30
1.00
1.25
1.50
1.75
2.00
2.25
2.50
Edge Density (Ratio of Edges to Possible Edges)
1e−5

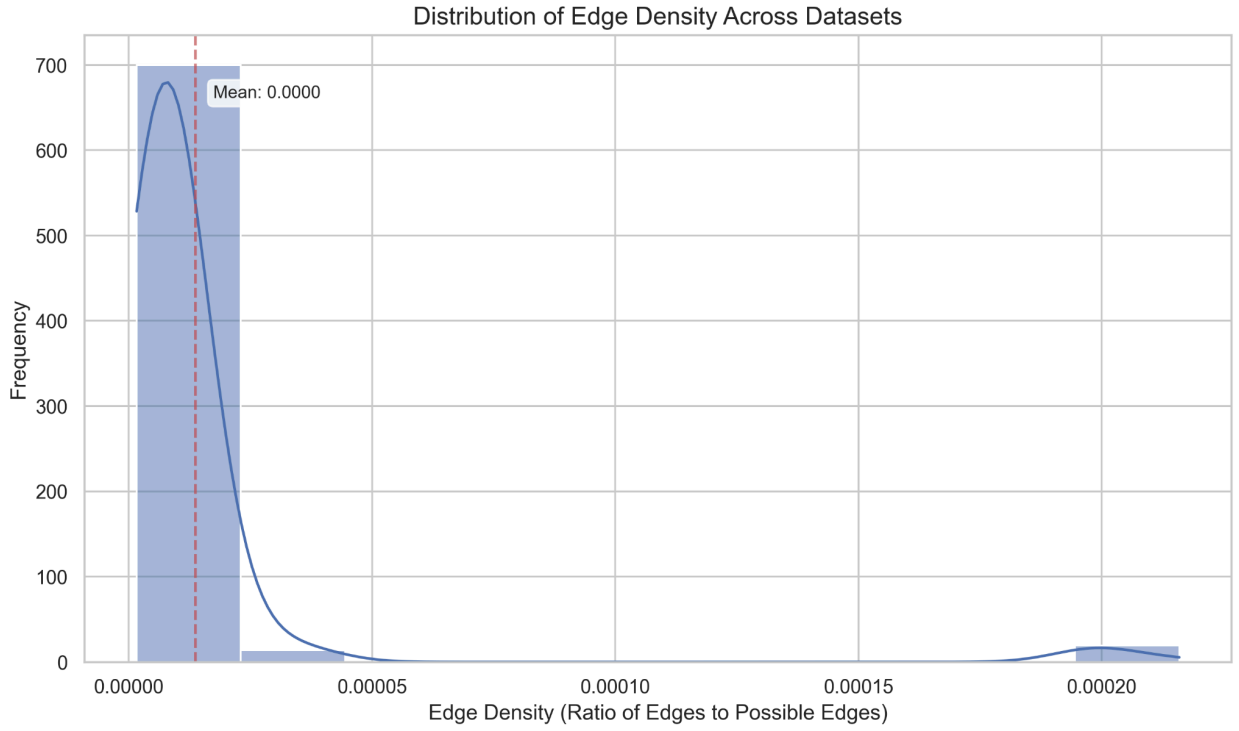
h=4



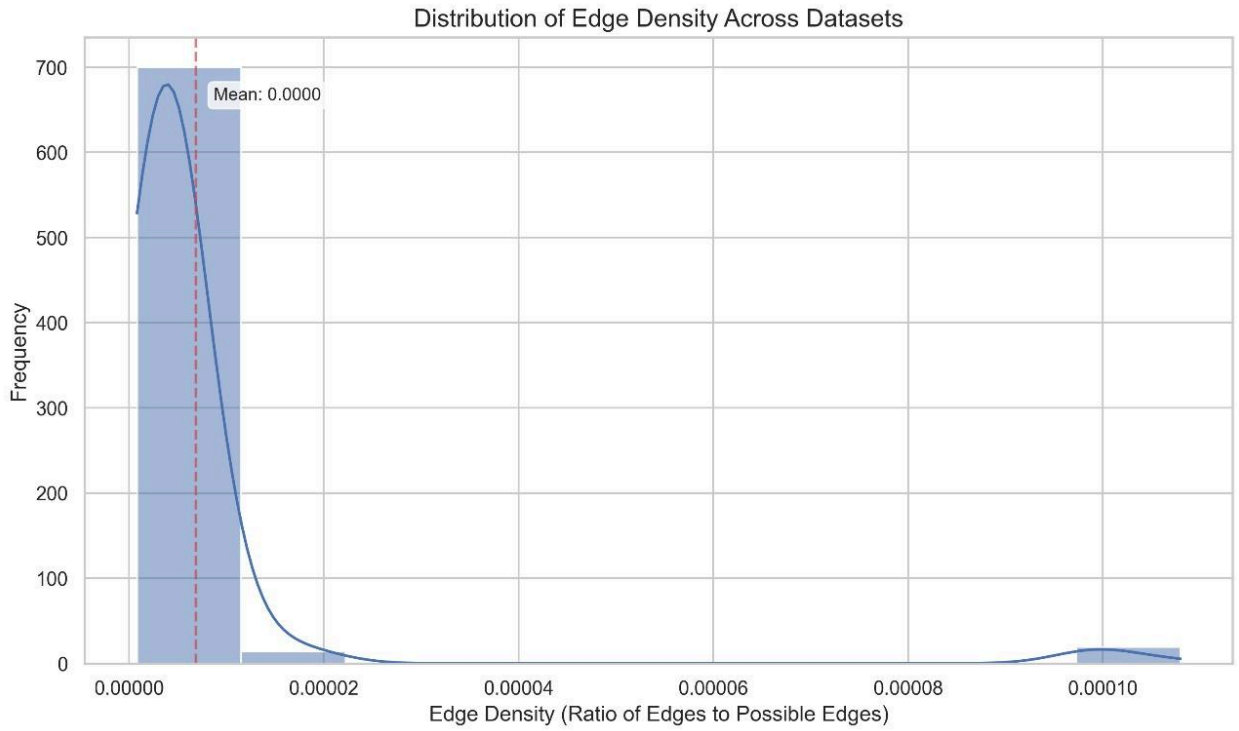
Dataset4 :

As-733

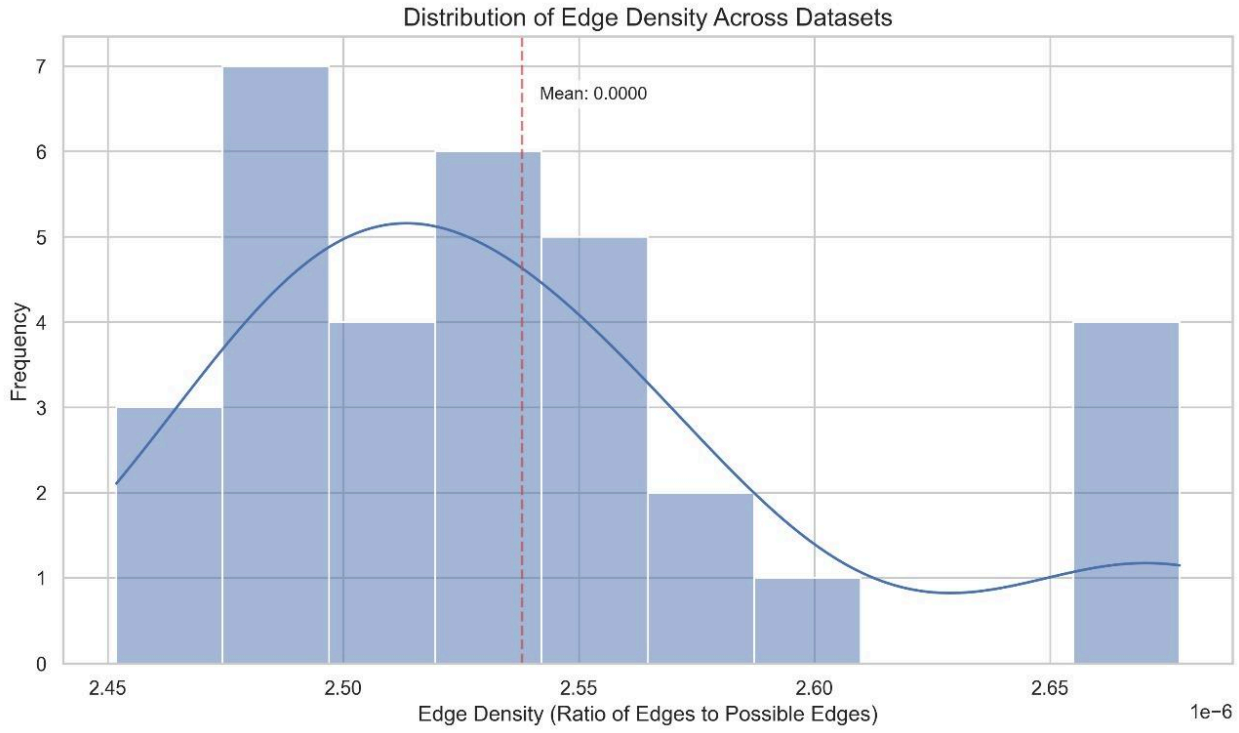
h=2



h=3



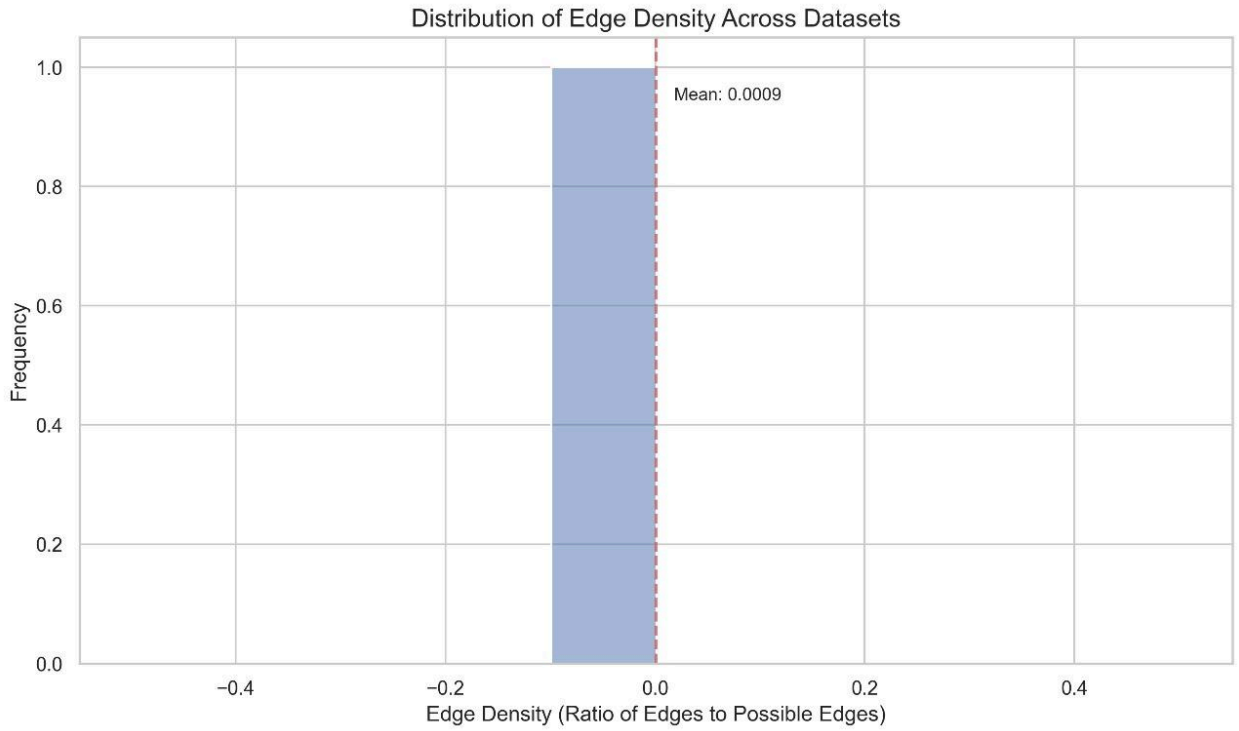
h=4



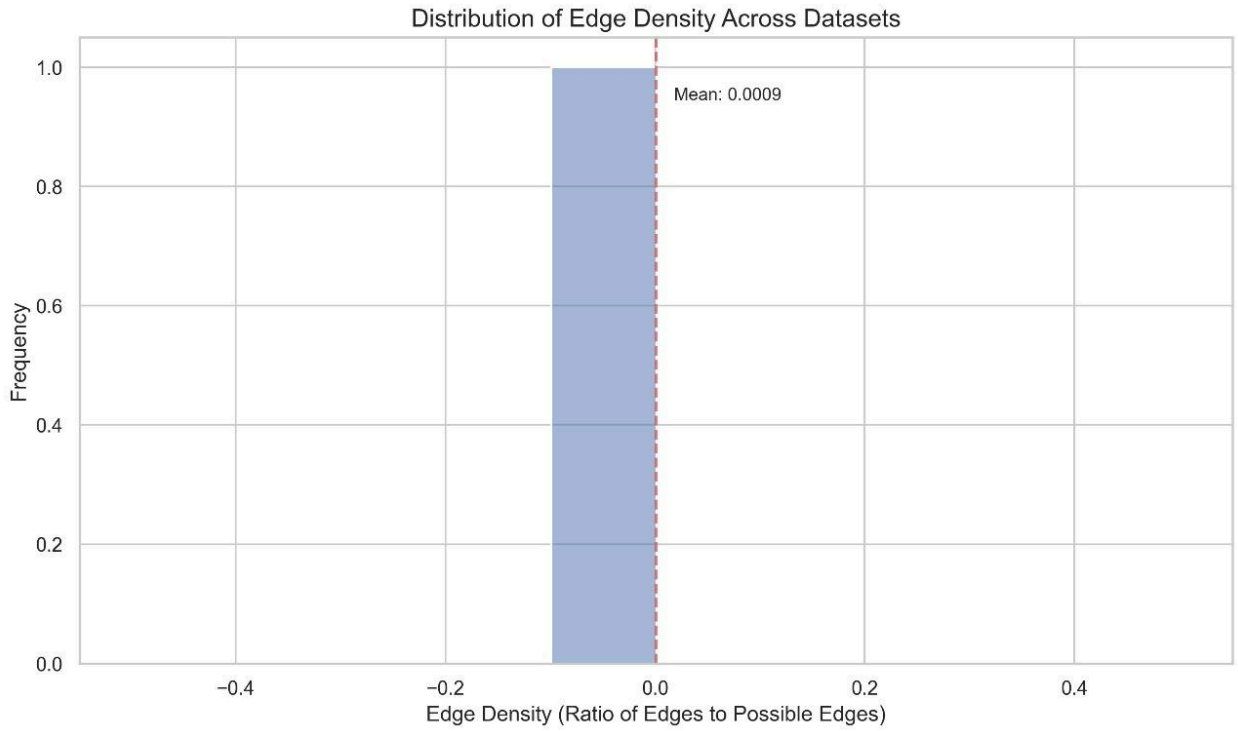
Dataset5 :

Yeast

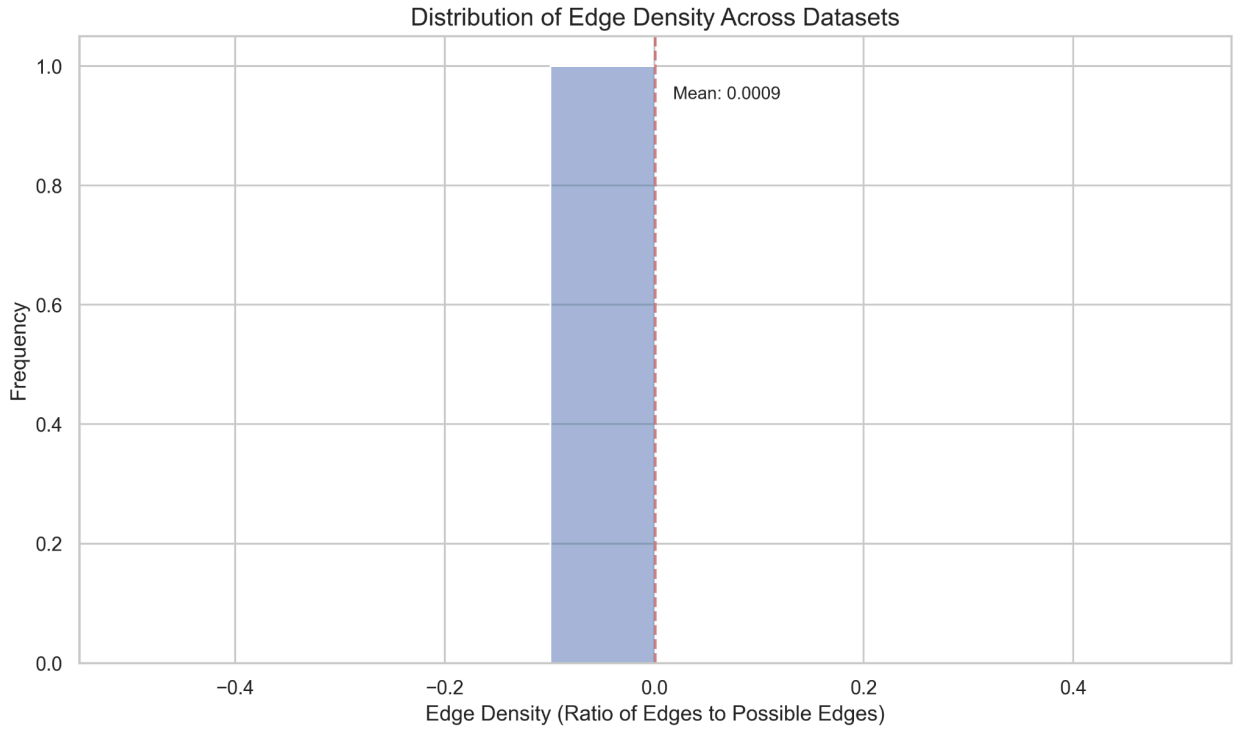
h=2


Distribution of Edge Density Across Datasets
Mean: 0.0009
Frequency
1.0
0.8
0.6
0.4
0.2
0.0
−0.4
−0.2
0.0
0.2
0.4
Edge Density (Ratio of Edges to Possible Edges)

h=3



h=4

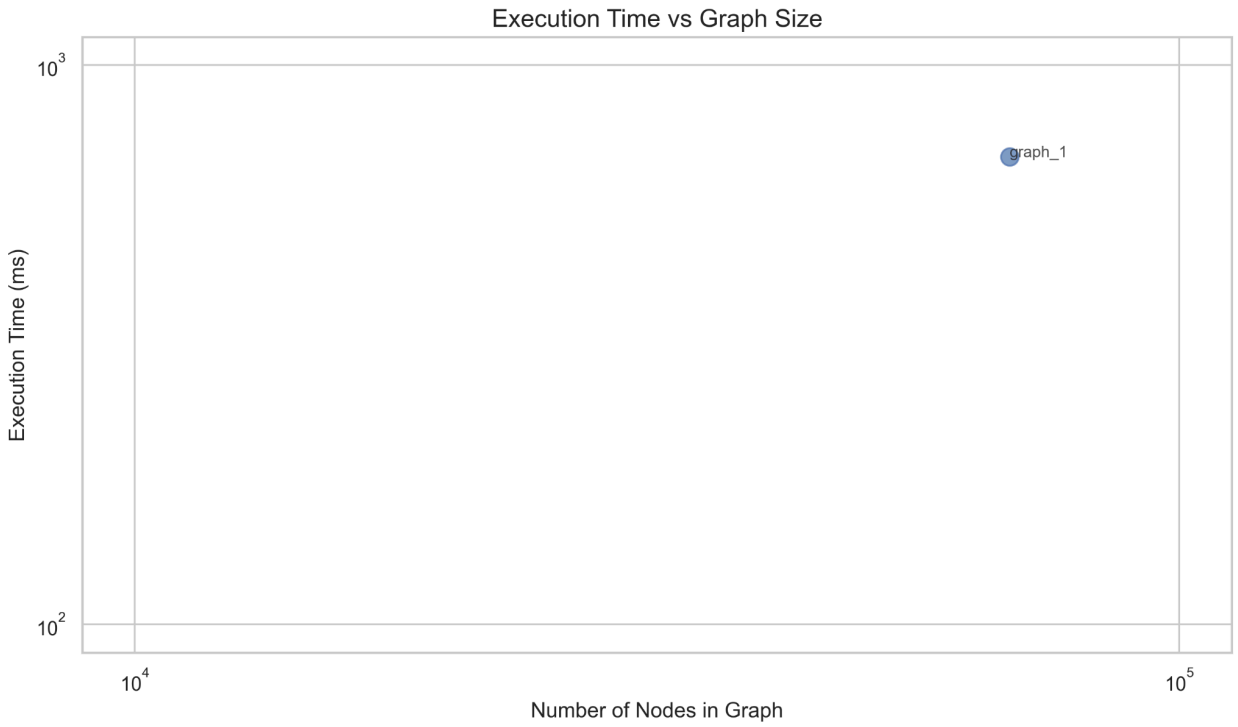


3.4 execution_time_vs_graph_size for Algorithm 4

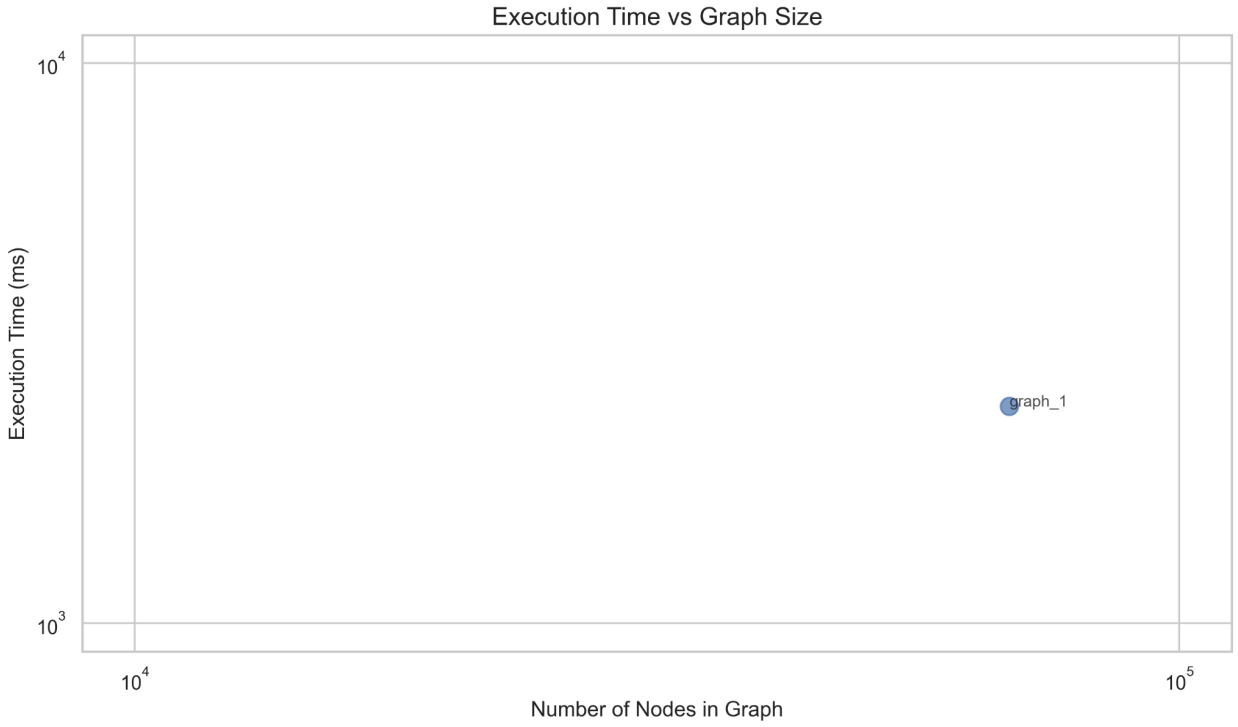
Dataset1 :

Ca-HepTh

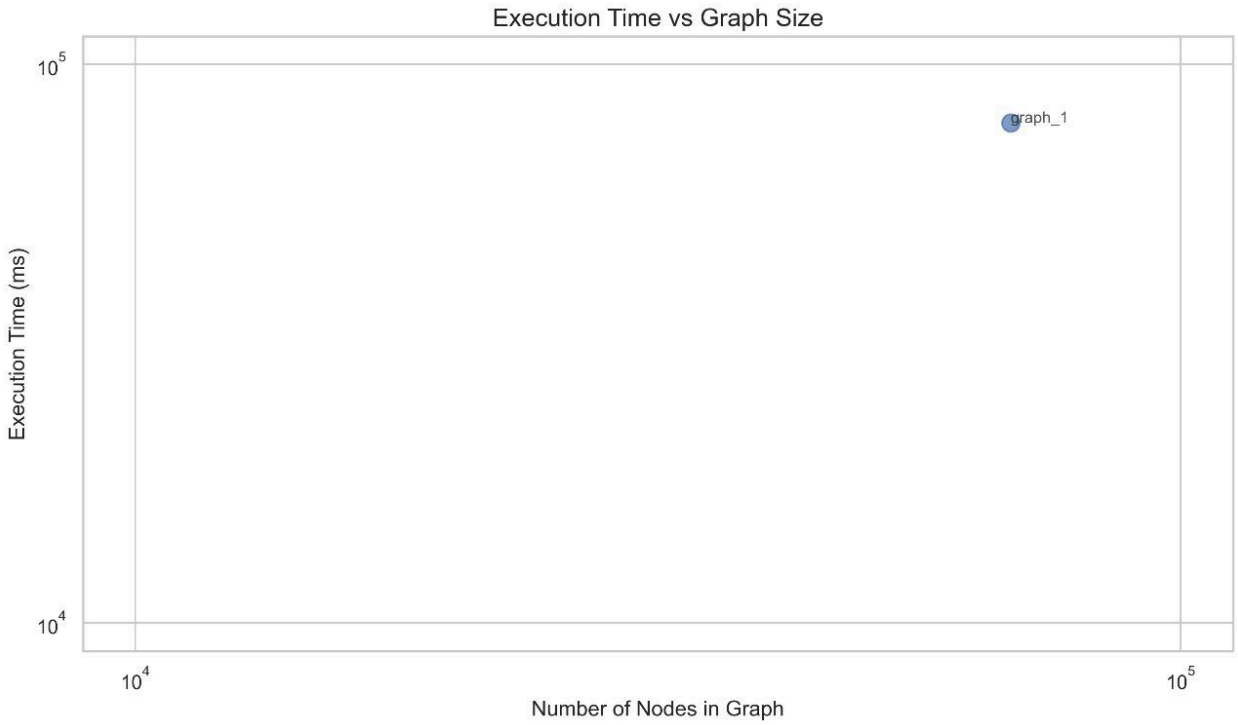
h=2


Execution Time vs Graph Size
10^3
10^2
graph_1
Execution Time (ms)
10^4
10^5
Number of Nodes in Graph

h=3


Execution Time vs Graph Size
10^4
10^3
graph_1
Execution Time (ms)
10^4
10^5
Number of Nodes in Graph

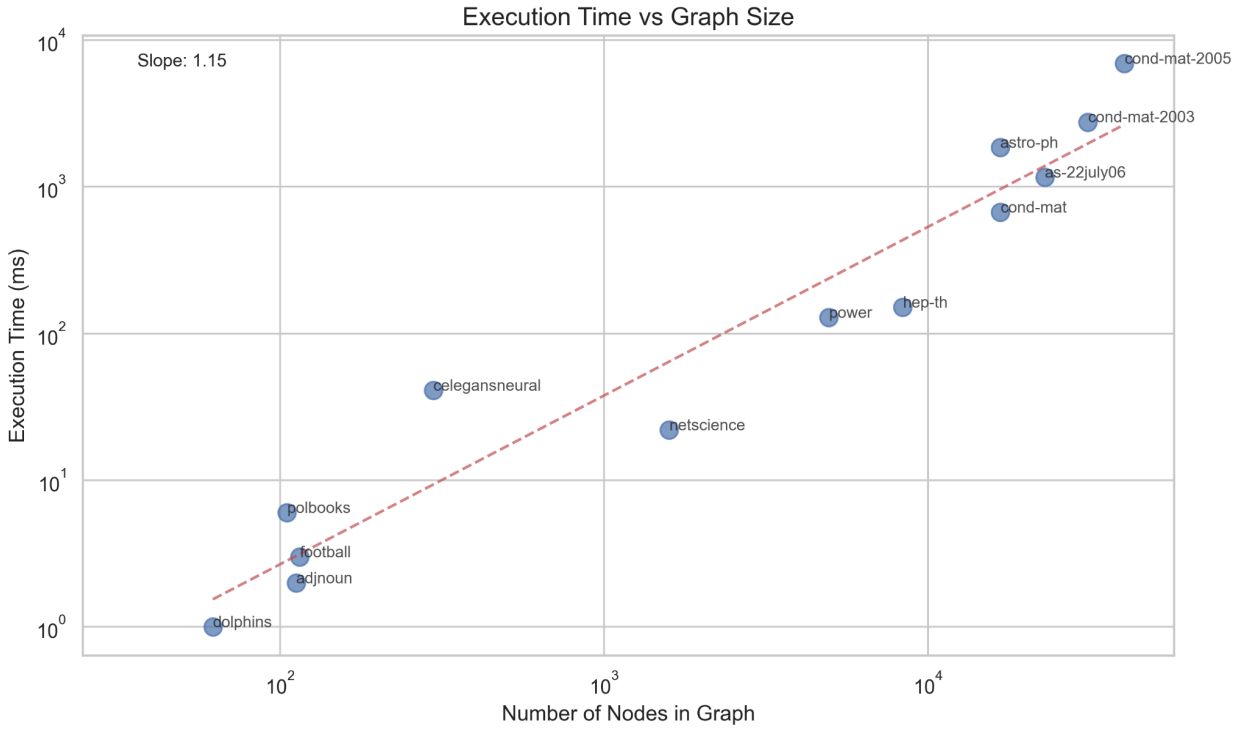
h=4



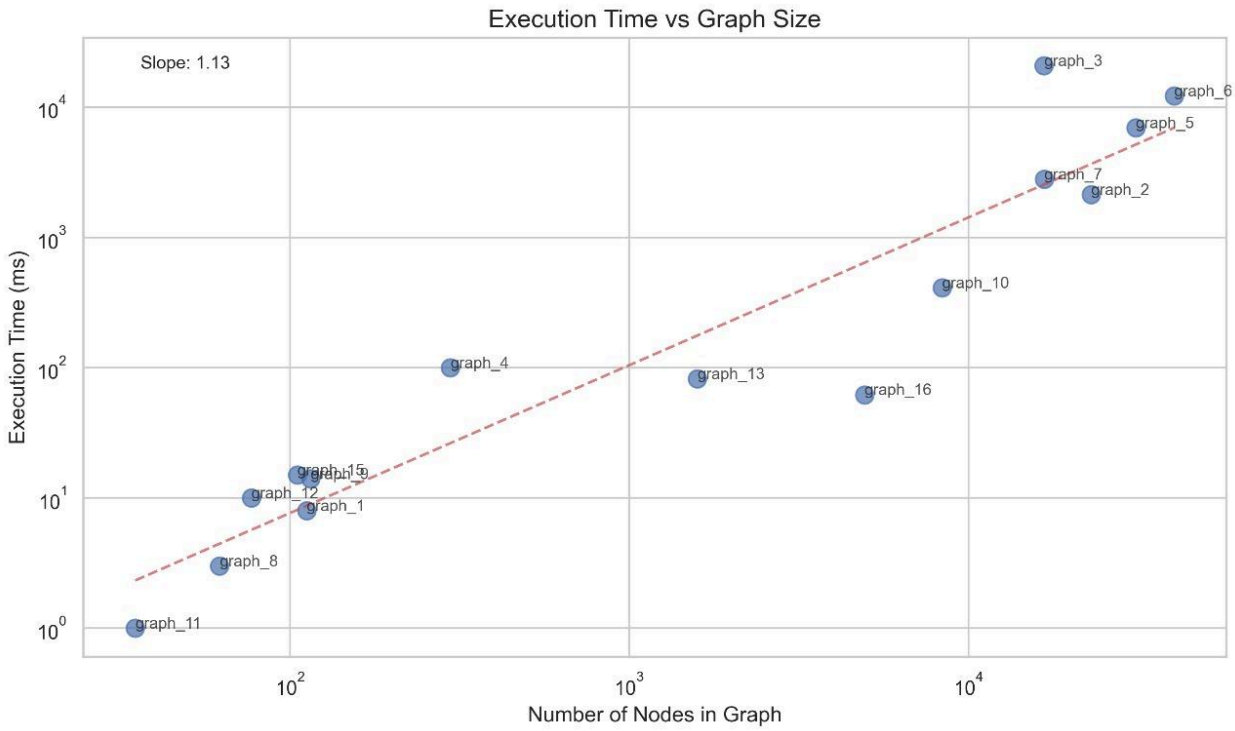
Dataset2 :

Netscience

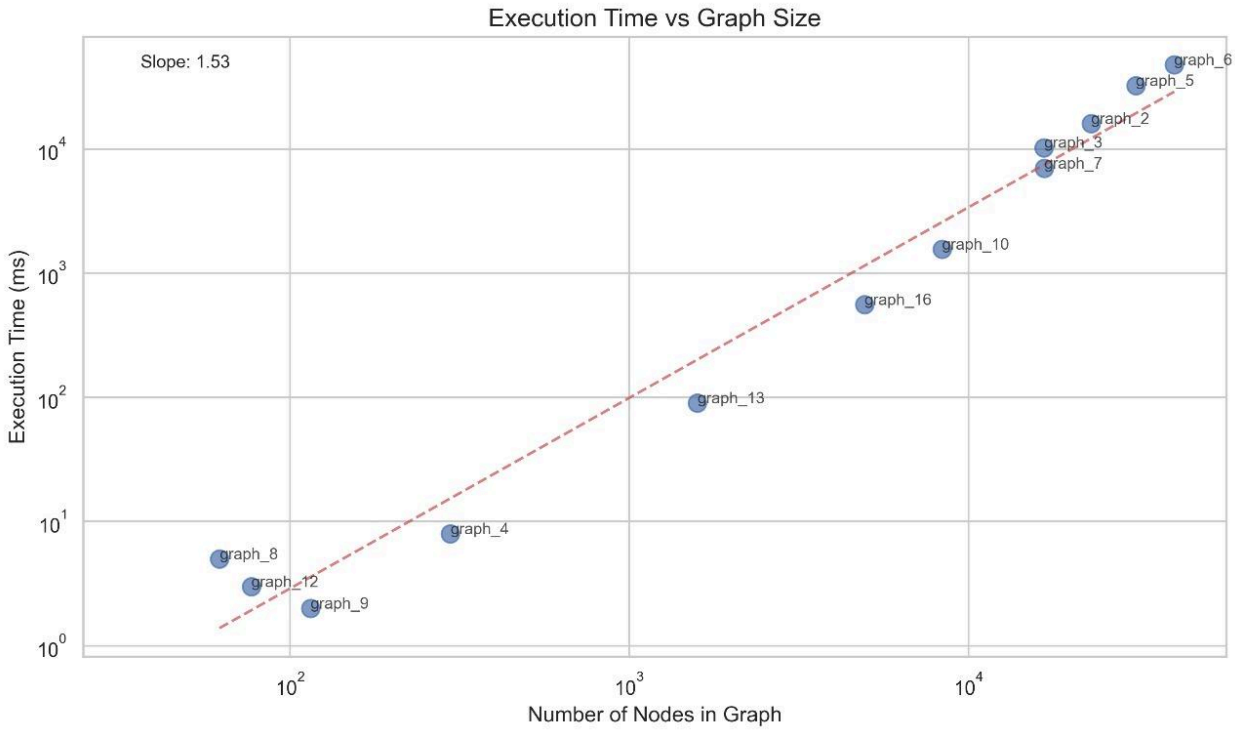
h=2



h=3



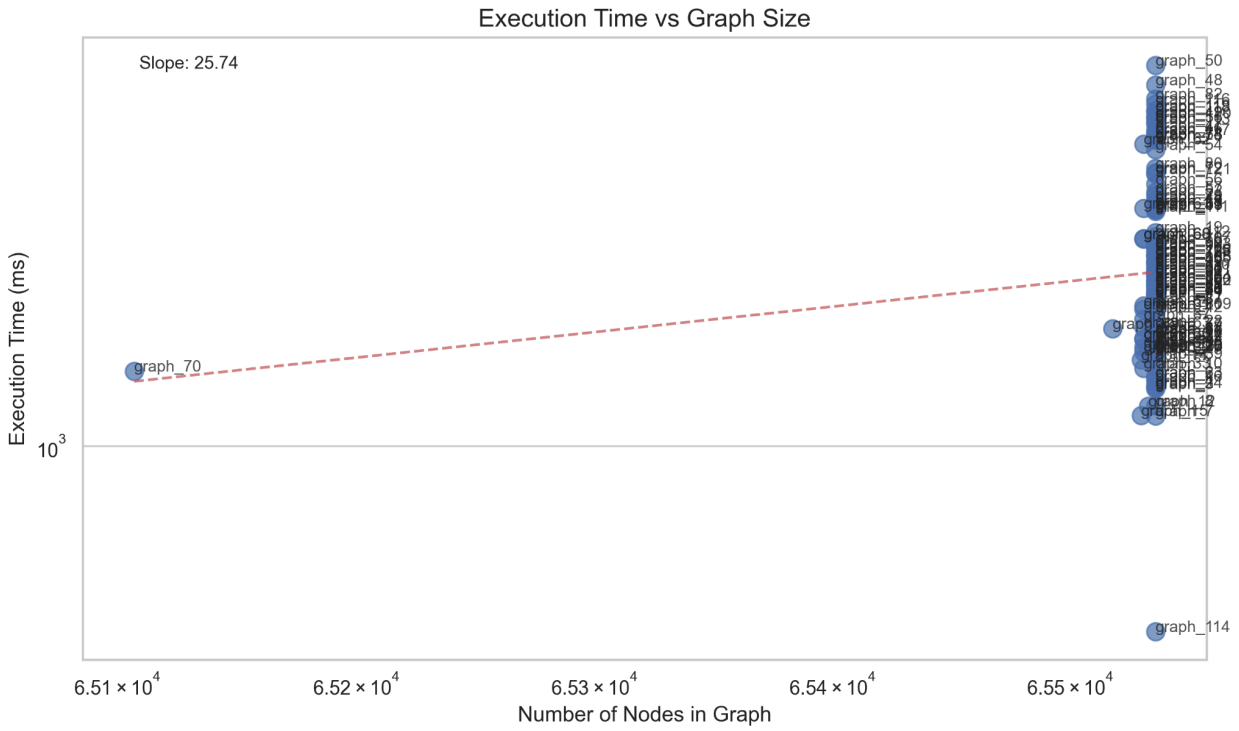
h=4



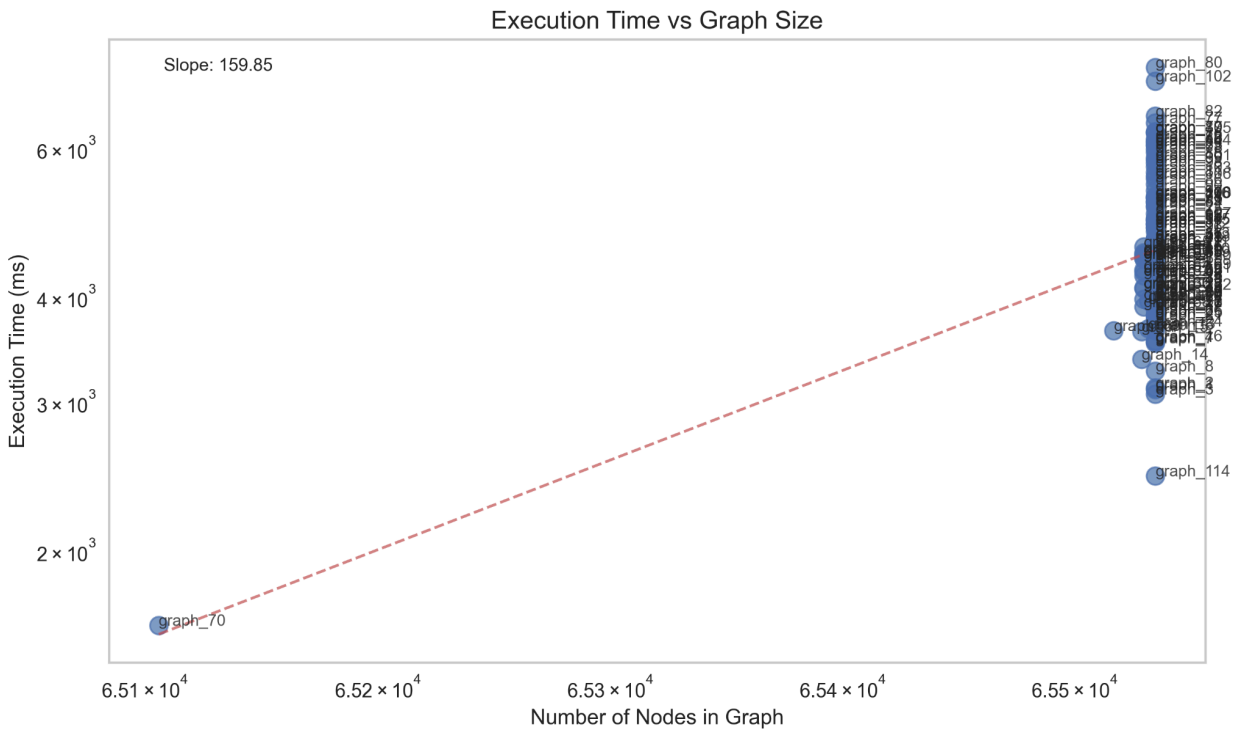
Dataset3 :

As-Caida

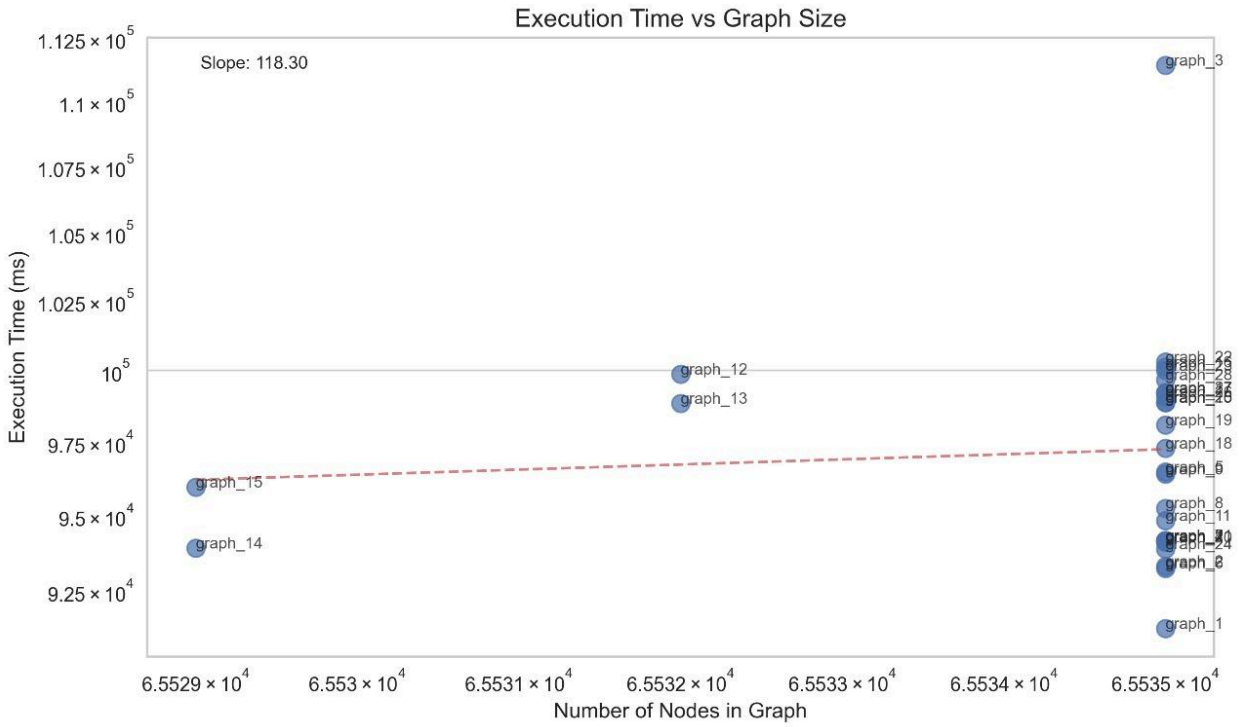
h=2


Execution Time vs Graph Size
Slope: 25.74
graph_50
graph_48
graph_70
graph_114
Execution Time (ms)
10^3
6.51×10^4
6.52×10^4
6.53×10^4
6.54×10^4
6.55×10^4
Number of Nodes in Graph

h=3


Execution Time vs Graph Size
Slope: 159.85
graph_80
graph_102
graph_14
graph_8
graph_114
graph_70
6×10^3
4×10^3
3×10^3
2×10^3
Execution Time (ms)
6.51×10^4
6.52×10^4
6.53×10^4
6.54×10^4
6.55×10^4
Number of Nodes in Graph

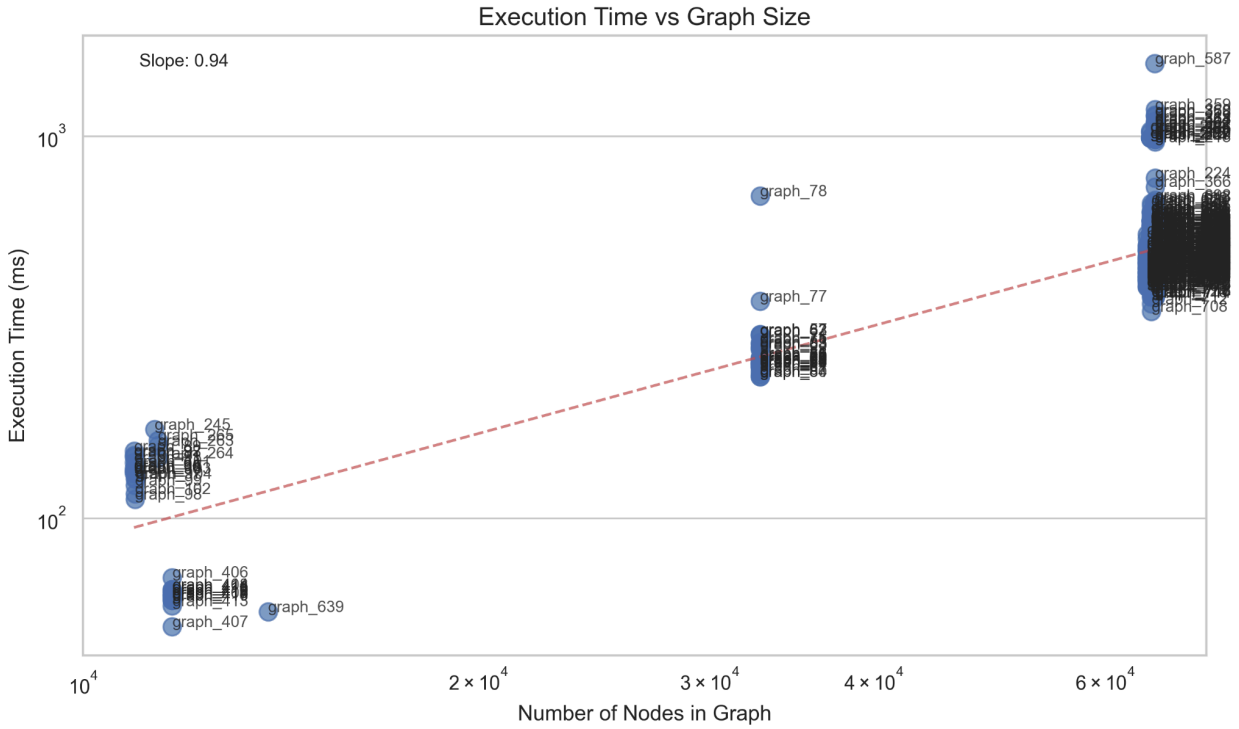
h=4



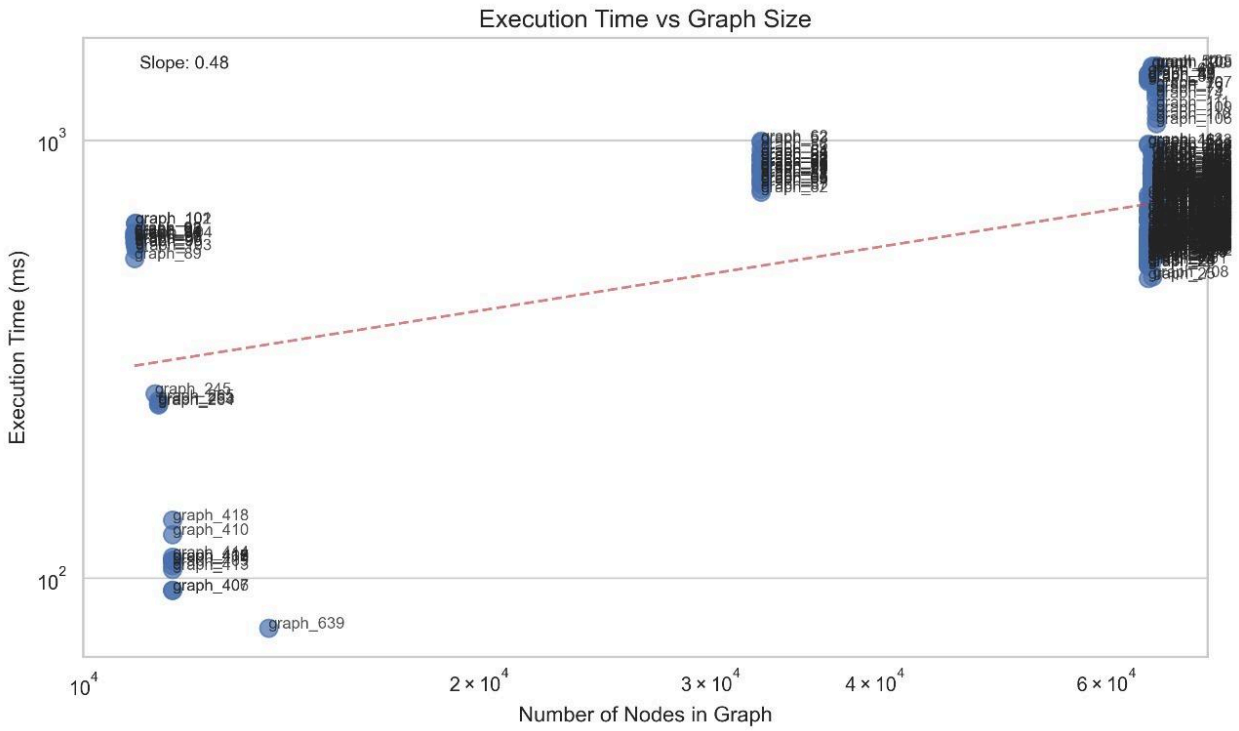
Dataset4 :

As-733

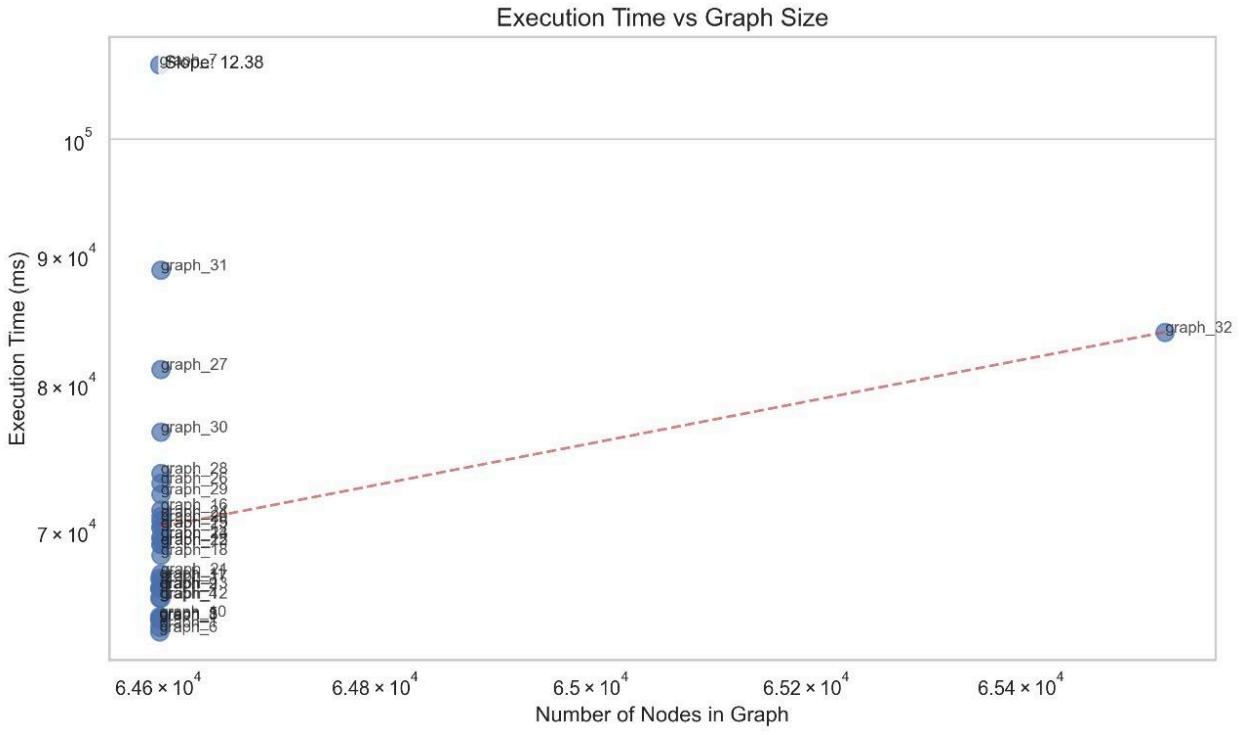
h=2



h=3



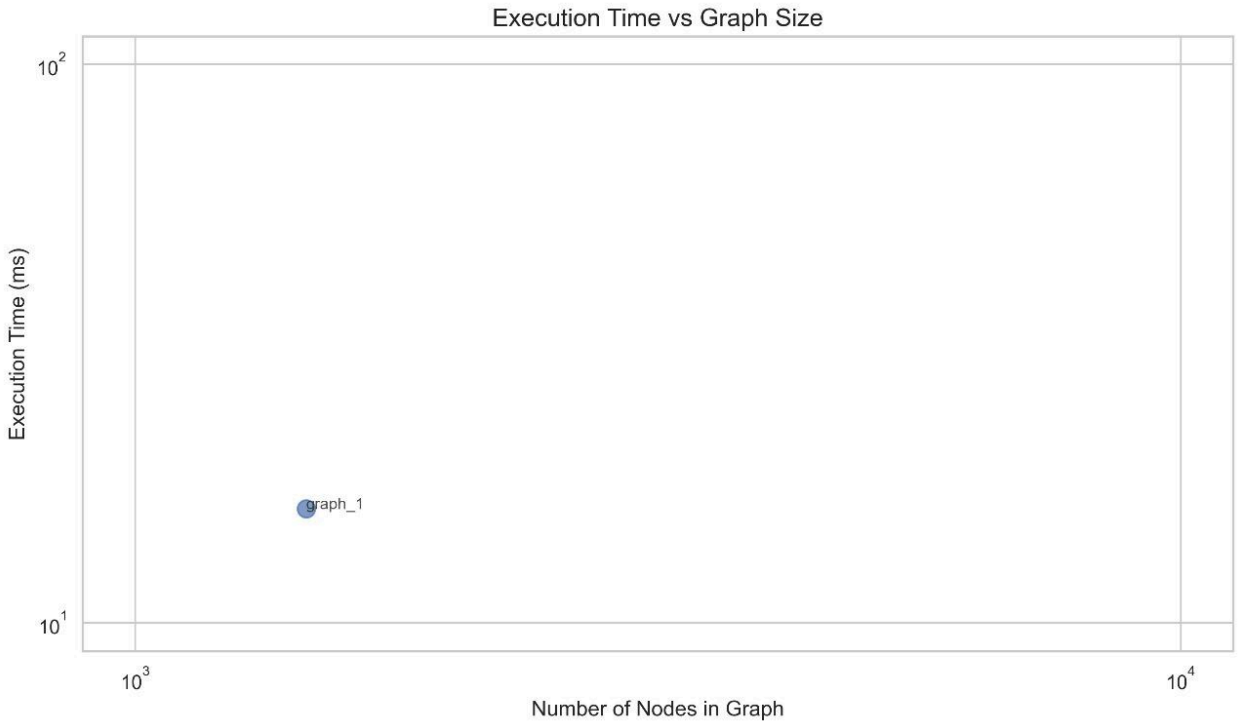
h=4



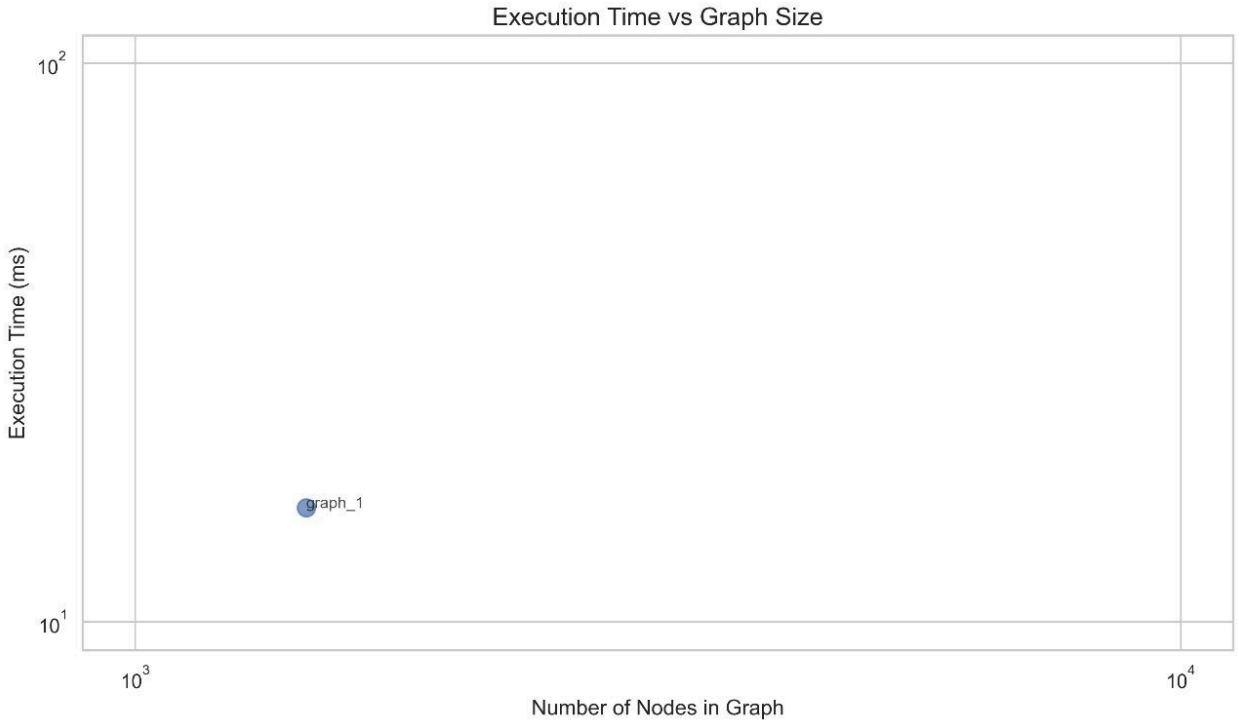
Dataset5 :

Yeast

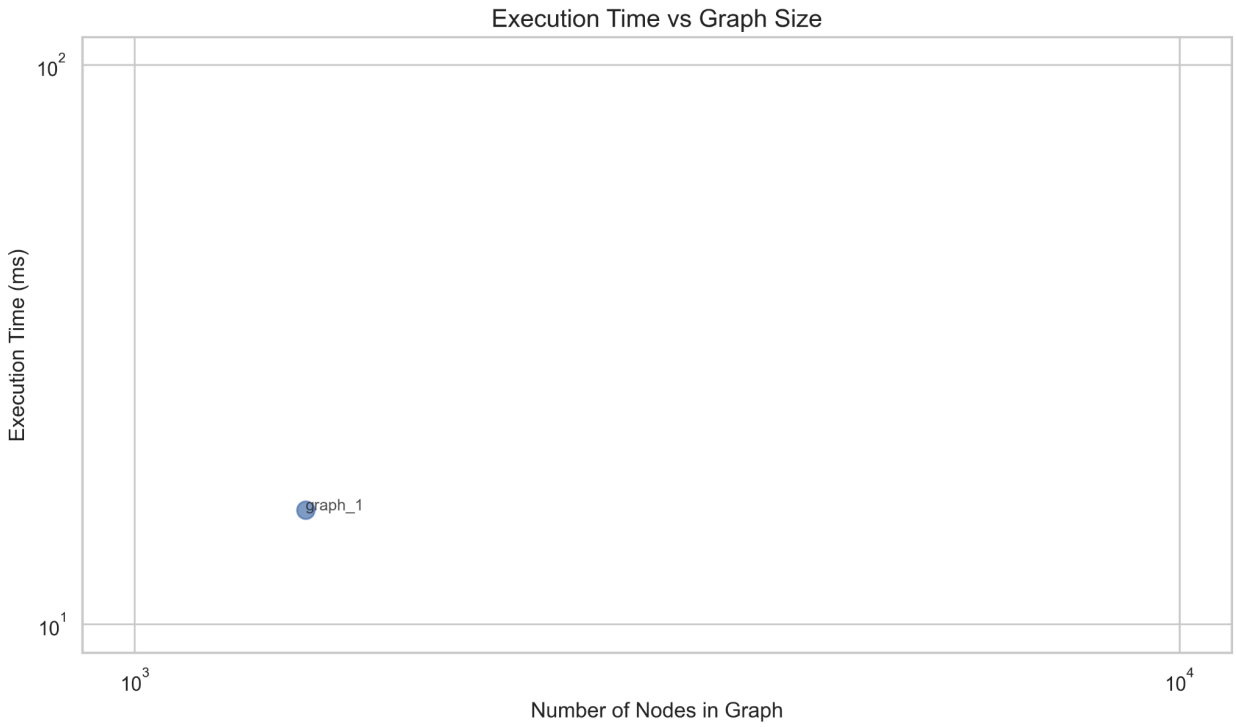
h=2


Execution Time vs Graph Size
10^2
10^1
Execution Time (ms)
graph_1
10^3
10^4
Number of Nodes in Graph

h=3


Execution Time vs Graph Size
10^2
10^1
Execution Time (ms)
graph_1
10^3
10^4
Number of Nodes in Graph

h=4


Execution Time vs Graph Size
10^2
10^1
Execution Time (ms)
graph_1
10^3
10^4
Number of Nodes in Graph

3.5 Observation :

3.5.1 Time Complexity Analysis :

The time complexity of the Exact algorithm (Algorithm 1) is dominated by its flow network operations. After initialization in O(n^h) time and flow network construction in O(|V| + |Λ|), the algorithm performs minimum cut computations for each (h-1)-clique, resulting in an overall complexity of O(n^(h-1)·|E|·f), where f represents the maximum flow value dependent on vertex degree relationships with parameter ψ. In comparison, CoreExact (Algorithm 4) achieves superior efficiency by incorporating strategic preprocessing. It begins with core decomposition in O(|V| + |E|) time, followed by component analysis of similar complexity. The key improvement comes from performing flow network operations only on identified core components rather than the entire graph, leading to a complexity of O(|V| + |E| + |C|²·|Ec|·fc). This approach significantly reduces computational overhead when the graph exhibits strong core structure, as is common in real-world sparse networks. The efficiency gains stem from CoreExact's ability to narrow the search space to promising regions, process components independently, and progressively refine solutions rather than solving the complete problem at once. These strategic optimizations make CoreExact particularly effective for large graphs where direct application of flow techniques would be prohibitively expensive.

4. REAL-WORLD APPLICATION

4.1 Social Network Analysis

The Exact and CoreExact algorithms can identify influential communities in social networks that may not be apparent through traditional clustering. These algorithms excel at detecting densely connected subgroups within larger, sparser networks—particularly useful for platforms like Reddit or Discord where communities may have intensely active core members surrounded by more casual participants.

4.2 Financial Networks

In financial markets, these algorithms can detect unusual trading patterns by identifying densely connected networks of transactions that might indicate market manipulation or insider trading. CoreExact's efficiency with large datasets makes it particularly valuable for analyzing high-frequency trading networks where traditional methods might be computationally prohibitive.

4.3 Supply Chain Optimization

In complex supply networks, these algorithms can identify critical supplier clusters where disruptions would have cascading effects. By mapping supplier relationships as graphs and applying CoreExact, companies can identify vulnerabilities where multiple important components share common suppliers, helping prioritize redundancy investments.

4.4 Recommendation Systems

E-commerce platforms can use these algorithms to identify dense subgraphs of frequently co-purchased items, revealing complementary product relationships that might not appear in traditional recommendation approaches. The density parameter ψ can be tuned to control the tightness of these product clusters.

4.5 Epidemiology and Disease Spread

In contact tracing applications, these algorithms can identify potential superspreader events by finding dense interaction clusters in mobility or contact data. CoreExact's ability to work with large sparse networks makes it suitable for analyzing population-scale contact networks where only specific regions exhibit high interaction density.

4.6 Network Security

These algorithms can detect botnet command and control structures by identifying abnormally dense communication patterns within larger network traffic. The flow network approach is particularly effective at finding coordination points where multiple compromised systems interface with control nodes.

5. CONCLUSION

In this assignment, we also explored two important algorithms for solving the Clique Densest Subgraph (CDS) problem. The **Exact** algorithm constructs a flow network to identify dense subgraphs by leveraging minimum cut techniques. It initializes with all (h-1)-cliques and systematically builds a flow network connecting vertices and cliques with capacities based on degree parameters. By finding minimum s-t cuts, it effectively identifies subgraphs that maximize density according to the parameter ψ, although this approach can be computationally intensive for larger graphs with a time complexity of $O(n^{(h-1)} \cdot |E| \cdot f)$. The **CoreExact** algorithm significantly improves upon this foundation by incorporating core decomposition as a preprocessing step. It first identifies promising dense regions through core decomposition, focuses only on sufficiently large connected components, and then applies flow network techniques selectively to these regions. This strategic reduction of the search space results in a more efficient time complexity of $O(|V| + |E| + |C|^2 \cdot |Ec| \cdot fc)$, making it particularly well-suited for large-scale real-world networks. Working with these algorithms demonstrated how structural preprocessing can dramatically improve computational efficiency when dealing with dense subgraph detection problems, which has important applications in community detection, network analysis, and biological network interpretation.