**Assignment 1** **Report**

FOR

**CS F364 DESIGN AND ANALYSIS OF ALGORITHM**

BY

GROUP 23

**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**

**(MARCH 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.  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
**TABLE OF CONTENTS**

1. Introduction……………………………………………………………………4

1.1 What is a Clique ?.................................................................................................................................4

1.2 What is Maximal Clique ?....................................................................................................................4

1.3 Decision Problem…………………………………………………………………………………….4

1.4 Optimization Problem………………………………………………………………………………..4

2. Exploring Research Papers……………………………………………....…….5

2.1 Listing All Maximal Cliques in Sparse Graphs in Near-optimal Time………………….…………...5

2.2 The worst-case time complexity for generating all maximal cliques and computational experiments…………………………………………………………………………………………..…...7

2.3 Arboricity and Subgraph Listing Algorithms………………………………………………………...9

3. Implementation & Observation………………………………………………..11

3.1 Listing All Maximal Cliques in Sparse Graphs in Near-optimal Time………………………..…….11

3.2 The worst-case time complexity for generating all maximal cliques and computational experiments………………………………………………………………………………………………11

3.3 Arboricity and Subgraph Listing Algorithms………………………………………………………..12

3.4 Running Time………………………………………………………………………………………...12

3.5 Observation…………………………………………………………………………………………...13

4. Real-World Application…………………………………………………….…14

5. Conclusion…………………………………………………………………….15

6. References…………………………………………………………………….16

**1.INTRODUCTION**

**1.1 What is a Clique ?**

If a subgraph of an undirected graph is a complete graph, it is called a clique. That is, a clique is a subgraph where every two vertices are neighbours, all nodes are directly connected with an edge. Cliques play a crucial role in various real-world applications, such as social network analysis, bioinformatics, and computer science. The study of cliques is very important as it helps in understanding and the research of network structures and relationships.

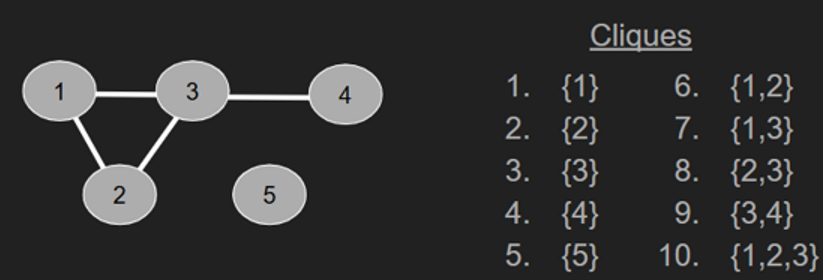


Fig 1.1 Clique Example

**1.2 What is maximal Clique ?**

A **maximal clique** in a graph is a subset of vertices that is completely connected and cannot be made any larger.In the above example (Fig 1.1) {5},{1,2,3},{3,4} are maximal cliques.

**1.3 Decision problem :**

It is a problem that can be posed as a yes-or-no question and the goal is to determine whether a given instance satisfies a particular property. For Example : Graph coloring problem, Subset Sum Problem, Satisfiablility Problem, etc.

Now the clique decision problem involves determining whether a graph contains a k clique or not. It is classified as NP-complete, i.e computationally challenging and no polynomial-time algorithm exists for solving it.

**1.4 Optimization Problem :**

It involves finding the best solution from a set of feasible solutions and the goal is to maximize an objective function. For Example : Traveling Saleman Problem, Knapsack Problem, Linear Programming.

Now the clique optimization problem refers to finding the largest or most optimal clique in a graph, depending on the specific criteria being maximized or minimized.

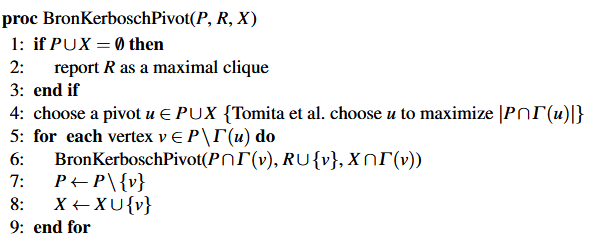
Maximum Clique Problem is NP-Hard.

**2. EXPLORING RESEARCH PAPERS**

**2.1 Listing All Maximal Cliques in Sparse Graphs in Near-optimal Time**

(David Eppstein, Maarten L¨offler, and Darren Strash)

Problem Statement : Given an undirected graph G = (V, E), find all maximal cliques in G.  
  
We are using the Bron-Kerbosch Algorithm with pivot to solve the problem. Firstly, before that, we have to understand the Bron-Kerbosch Algorithm without pivoting. Basically, it takes three inputs: the current clique, the candidate set, and the exclusion set. The current clique helps to build up a clique one vertex at a time. The candidate set consists of vertices that we could potentially add to the current clique to make it a larger clique, and the exclusion set consists of vertices that we have already processed. If they are adjacent to the current clique, then it means the current clique is not maximal. We are using backtracking and recursive calls. Backtracking helps once we get to a state where we cannot find the clique; it backtracks out and goes to the previous state.



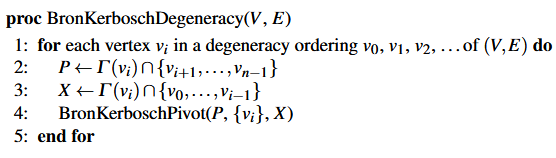


Fig 2 : Bron-Kerbosch Algorithm

At a time, the search space has three inputs: initially, the current clique and the current set are empty, and the candidate set has all the vertices. Firstly, we check for the base case. If the candidate set and exclusion set are not empty, then we go to the recursive function call, in which we add a vertex and then see the common current set and the neighborhood of it, for the candidate set and for the exclusion set, the intersection of the neighborhood of the vertices, which would be empty initially. The exclusion set is empty initially, and we keep calling the recursion function until the base case hits when the candidate and exclusion set are empty. The current clique is printed, and then we backtrack, and vertices are removed if they are already processed. So, in a future iteration, they are no longer in the candidate set because they are already processed, and we are adding them to the exclusion set as they are processed so that any result we get in the future, we want to make sure it is not adjacent to those vertices. If we find a clique that is still adjacent to those vertices, then it means it is not the maximal clique.

So now, the previously explained approach was without using a pivot, which was causing more states. To optimize this, we will use pivoting. The idea is that rather than iterating at every vertex in the candidate set at every recursive step, instead, we are going to select a pivot vertex, u here the selection of pivot element is selected such that it is any vertex from candidate or exclusion set ., and iterate over a smaller set. That smaller set is going to be the candidate set minus the neighborhood of u. The idea behind this is that any clique that exists in the graph must contain vertex u or a non-neighbor of u, and if a clique does not contain a non-neighbor of u, then u could be added to the clique to make it larger.

**2.2 The worst-case time complexity for generating all maximal cliques and computational experiments** (Etsuji Tomita, Akira Tanaka, Haruhisa Takahashi)

We are using Tomita's Algorithm to solve the problem. Firstly, before that, we have to understand the Bron-Kerbosch Algorithm without pivoting. Basically, it takes three inputs: the current clique, the candidate set, and the exclusion set. The current clique helps to build up a clique one vertex at a time. The candidate set consists of vertices that we could potentially add to the current clique to make it a larger clique, and the exclusion set consists of vertices that we have already processed. If they are adjacent to the current clique, then it means the current clique is not maximal. We are using backtracking and recursive calls. Backtracking helps once we get to a state where we cannot find the clique; it backtracks out and goes to the previous state.

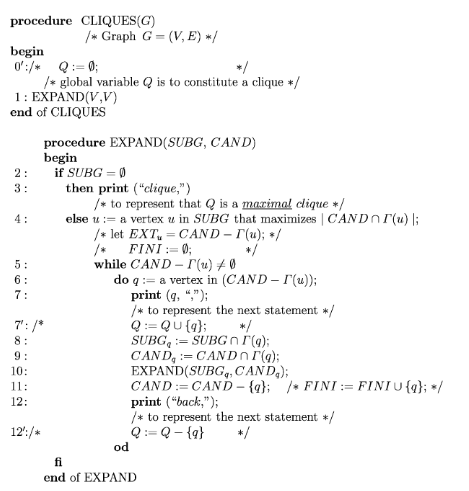


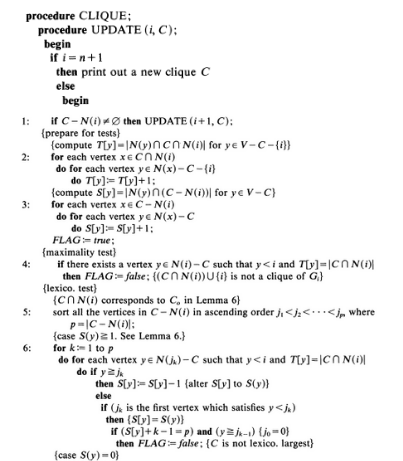
Fig 3 : Tomita’s Cliques Algorithm

At a time, the search space has three inputs: initially, the current clique and the current set are empty, and the candidate set has all the vertices. Firstly, we check for the base case. If the candidate set and exclusion set are not empty, then we go to the recursive function call, in which we add a vertex and then see the common current set and the neighborhood of it, for the candidate set and for the exclusion set, the intersection of the neighborhood of the vertices, which would be empty initially. The exclusion set is empty initially, and we keep calling the recursion function until the base case hits when the candidate and exclusion set are empty. The current clique is printed, and then we backtrack, and vertices are removed if they are already processed. So, in a future iteration, they are no longer in the candidate set because they are already processed, and we are adding them to the exclusion set as they are processed so that any result we get in the future, we want to make sure it is not adjacent to those vertices. If we find a clique that is still adjacent to those vertices, then it means it is not the maximal clique.

So now, the previously explained approach was without using a pivot, which was causing more states. To optimize this, we will use pivoting. The idea is that rather than iterating at every vertex in the candidate set at every recursive step, instead, we are going to select a pivot vertex, u here the selection of pivot element is selected such that it is a vertex such that it has the most neighbors in the candidate set . This helps minimize the number of vertices in the candidate set that are not neighbors of the pivot, which in turn reduces the number of recursive calls., and iterate over a smaller set. That smaller set is going to be the candidate set minus the neighborhood of u. The idea behind this is that any clique that exists in the graph must contain vertex u or a non-neighbor of u, and if a clique does not contain a non-neighbor of u, then u could be added to the clique to make it larger.

**2.3 Arboricity and Subgraph Algorithms** (NORISHIGE CHIBA, TAKAO NISHIZEKI)

The CLIQUE() algorithm efficiently lists all maximal cliques of a graph in a structured edge-searching strategy. It takes an undirected graph G(V, E) represented in an adjacency list format, where V is the set of vertices and E is the set of edges. The vertices are first sorted in non-decreasing order of degree to allow the algorithm to process lower-degree vertices first and gradually reduce the graph. The algorithm then employs recursive backtracking to construct cliques progressively, with only valid extensions and no repeated computation. In doing so, it performs two fundamental checks: the maximality check, to avoid adding a vertex so that it would not violate the definition of a clique, and the lexicographic check, to avoid generating duplicate cliques.



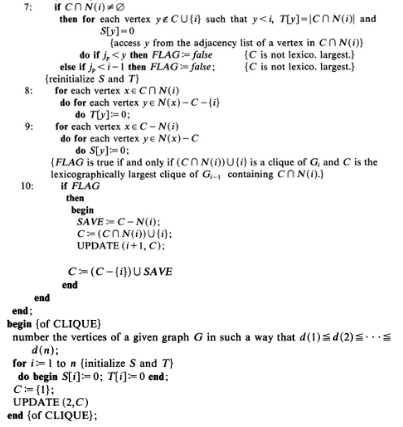


Fig 4

The algorithm utilizes some data structures including adjacency lists to store the graph, sets for maintaining in-current candidate cliques, arrays like S[] and T[] for maintaining temporary computation for verification of the clique, and stacks to manage recursion. A critical optimization is the usage of arboricity, a(G), that reduces the search space and contributes to efficiency, particularly for sparse graphs. The algorithm prints or stores in lists all of the maximal cliques of a graph, through degree-based ordering, recursion, optimizations of traversals of the edges, and backtracking performs efficiently an O(a(G) \* m) time per clique, which particularly works well in the case of large and sparse graphs.

**3. IMPLEMENTATION AND OBSERVATION**

We have implemented all 3 of the above algorithms and have included the code files on our github repository : <https://github.com/Kal-El-pt2/DAA_Assignment>.

Here are the results :

**3.1 The worst-case time complexity for generating all maximal cliques and computational experiments**

Dataset1 :

Wiki-Vote

Largest Clique Size: 17

Total Number of Maximal Cliques: 459002

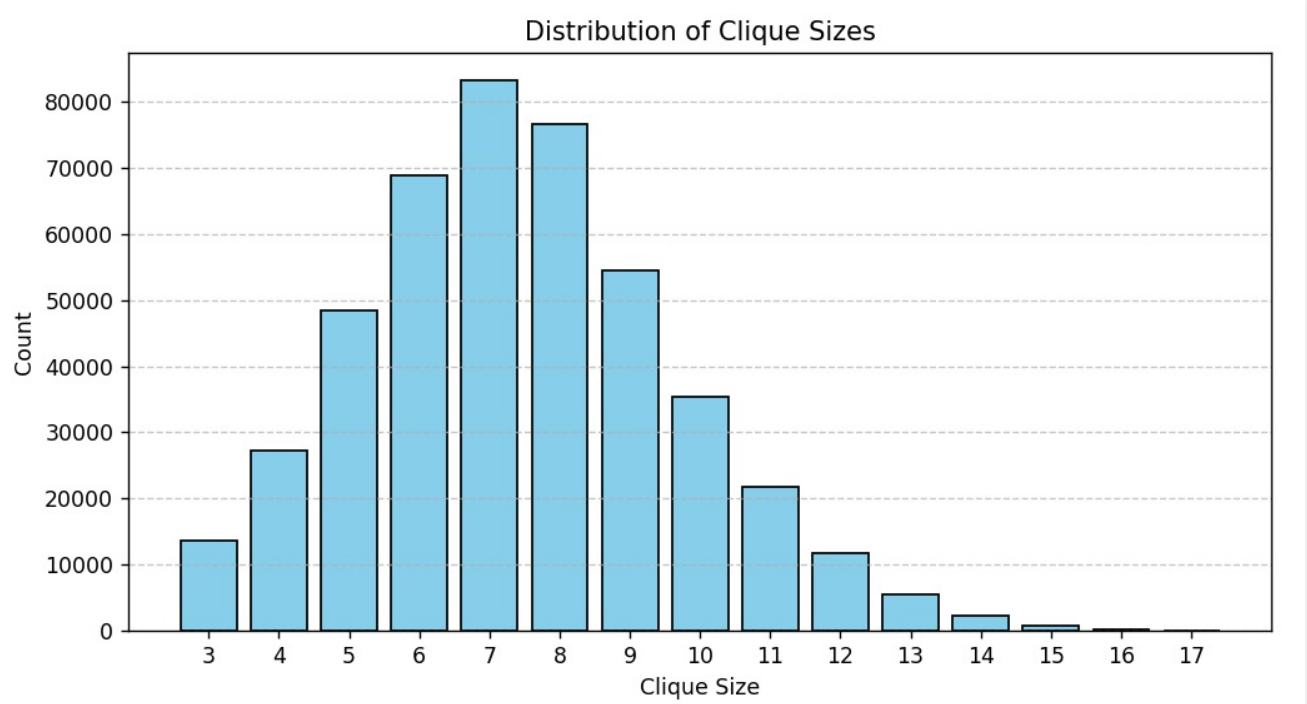


Fig 1

Dataset2 :

Enron-Email

Largest Clique Size: 20

Total Number of Maximal Cliques: 226859

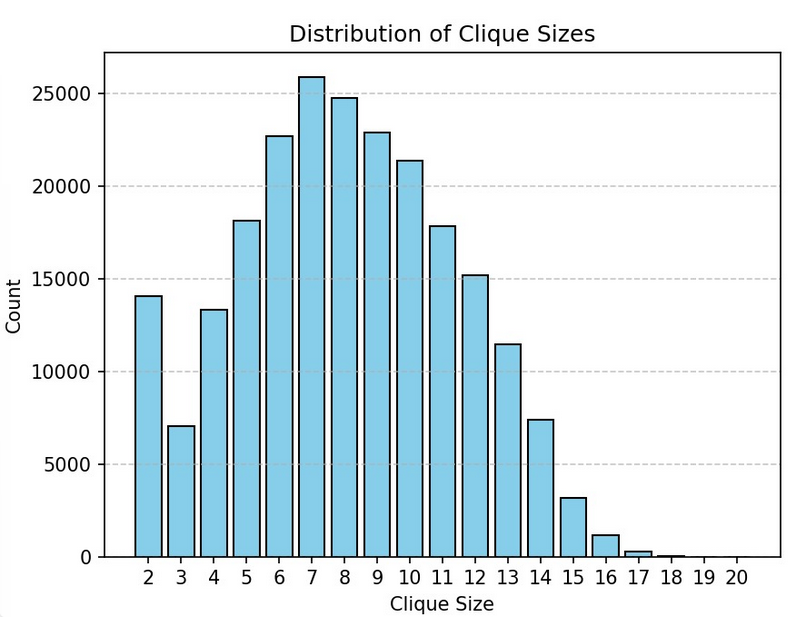


Fig 2

Dataset3 :

As-Skitter

Largest Clique Size: 67

Total Number of Maximal Cliques: 37322355

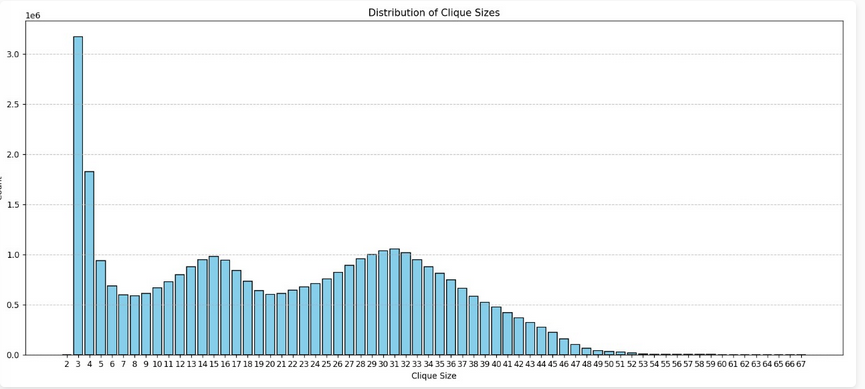


Fig 3

**3.2**  **Listing All Maximal Cliques in Sparse Graphs in Near-optimal Time**

Dataset1 :

Wiki-Vote

Largest Clique Size: 17

Total Number of Maximal Cliques: 459002

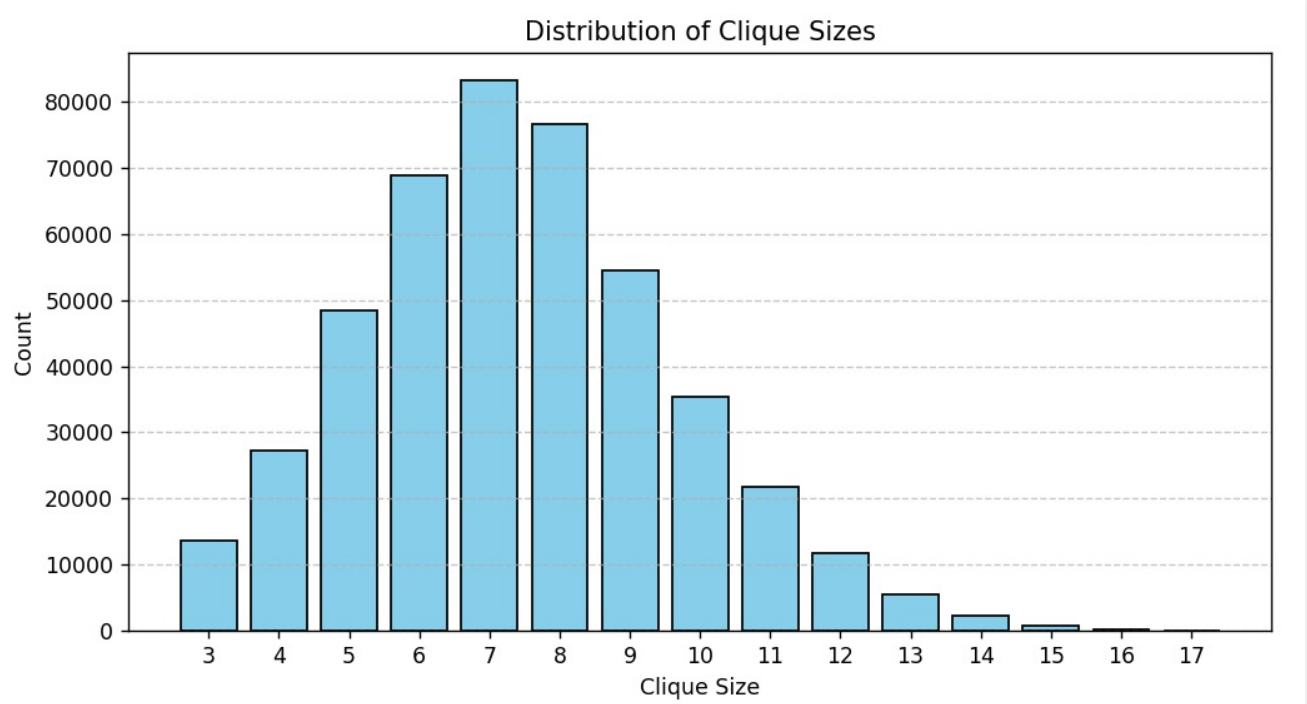


Fig 4

Dataset2 :

Enron-Email

Largest Clique Size: 20

Total Number of Maximal Cliques: 226859

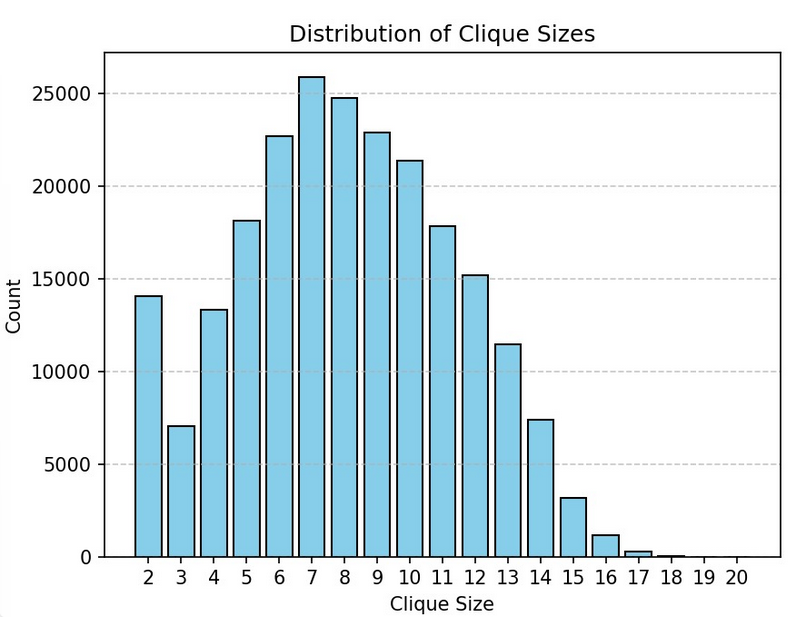


Fig 5

Dataset3 :

As-Skitter

Largest Clique Size: 67

Total Number of Maximal Cliques: 37322355

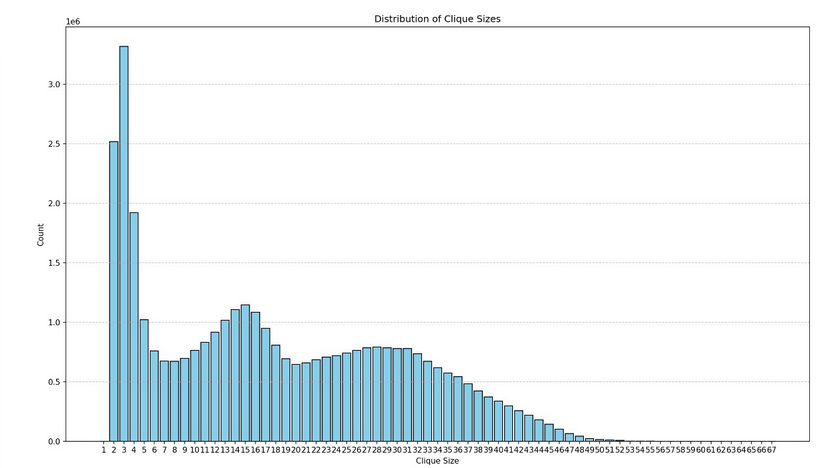


Fig 6

**3.3 Arboricity and Subgraph Algorithms**

Dataset1 :

Wiki-Vote

Largest Clique Size: 17

Total Number of Maximal Cliques: 459002

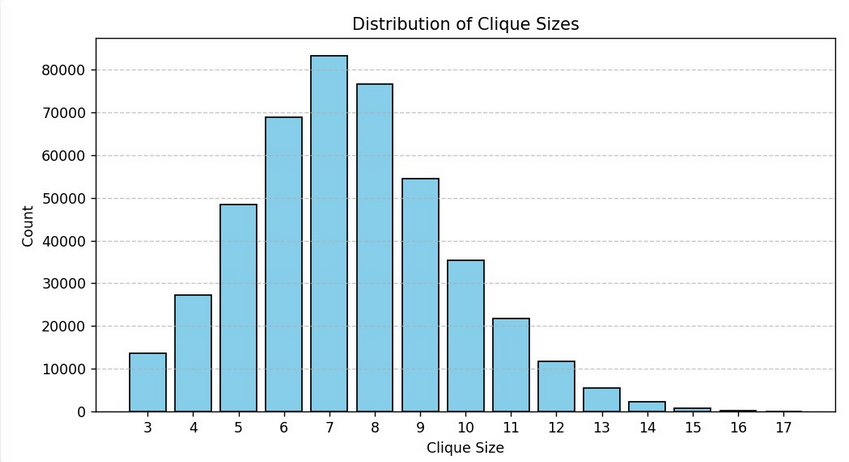


Fig5

Dataset2 :

Email-Eron

Largest Clique Size: 20

Total Number of Maximal Cliques: 226859

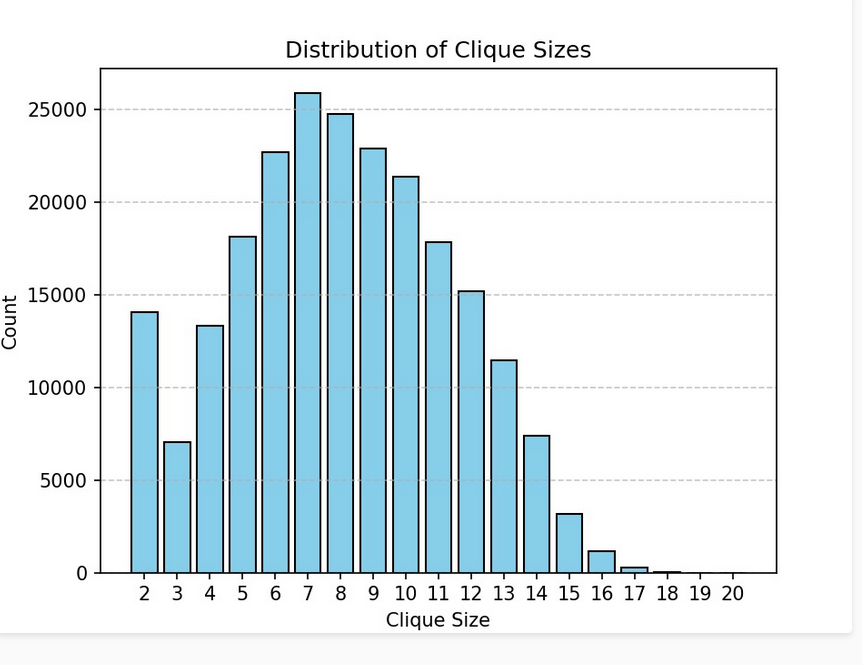


Fig 6

Dataset3 :

As-Skitter

Largest Clique Size: 67

Total Number of Maximal Cliques: 37322355

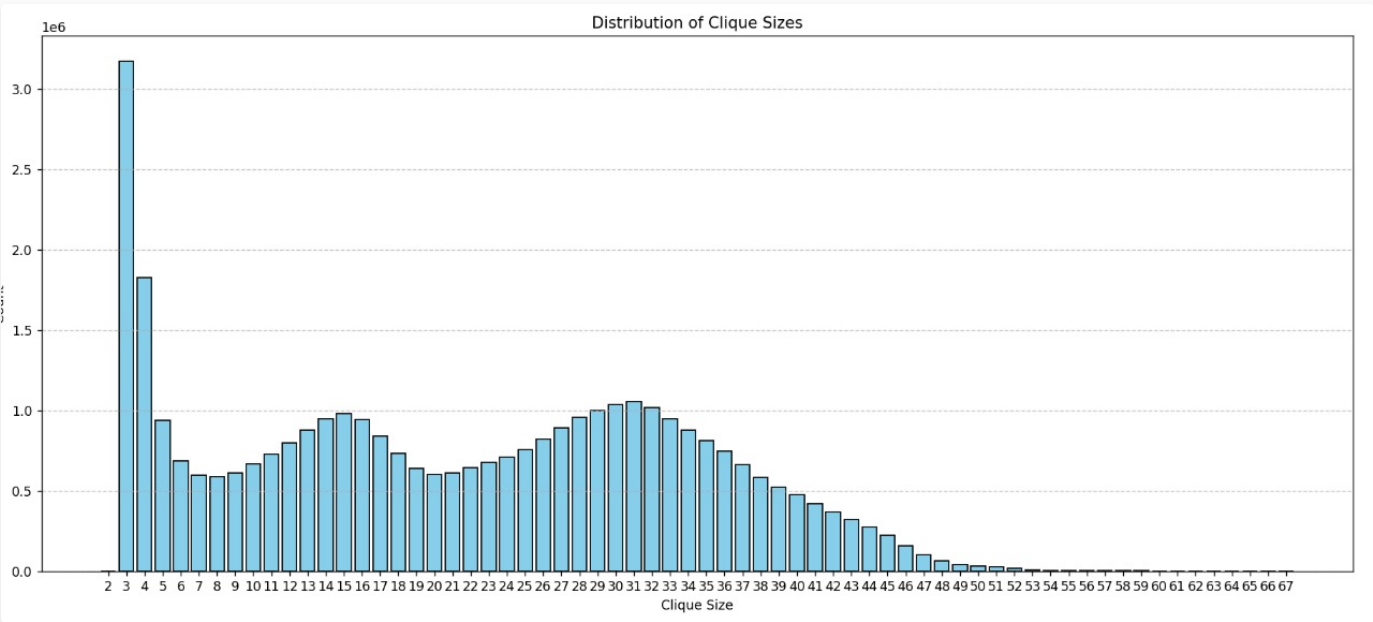


Fig 7

**3.4 Running Time :**

Dataset1 : Wiki-Vote.txt

Tomita : 2.03s

ELS : 14.89s

Chiba : 2min 7secs

Dataset2 : Email-Enron.txt

Tomita : 2.57s

ELS : 39.2s

Chiba : 3mins 10s

Dataset3 : as-skitter.txt

Tomita : 10mins

ELS : 31mins

Chiba : 12hrs 7mins

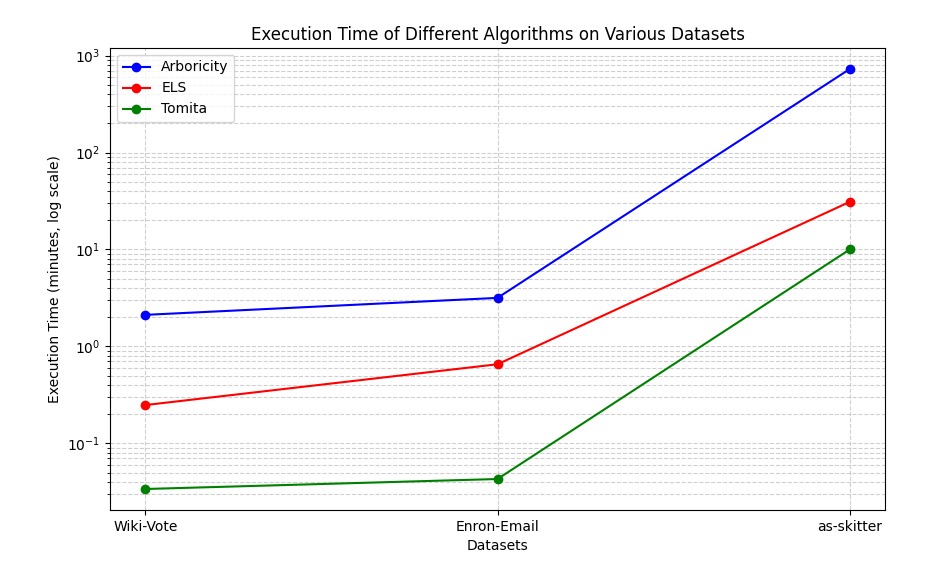


Fig 8

**3.5 Observation :**

**3.5.1 Time Complexity Analysis :**

The Bron-Kerbosch Algorithm had a pivoting technique that reduced unnecessary recursive calls and helped in optimizing the performance. It focuses on smaller candidate sets and due to this it demonstrates efficiency in sparse graphs. Then we have the Tomita’s algorithm that showed improvements over Bron-Kerbosch without pivoting by selecting vertices with maximum neighbors in the candidate sets, thus it managed to make recursive calls efficient, reducing computational overhead. Bron-Kerbosch has O(3^n/3) running time and Tomita has O(d\*3^n/3).

Then we have the Arboricity-based algorithm that uses graph sparsity to achieve better performance in edge dense regions. O(d(i)+Y,xcnN) d(x)).

**3.5.2 Dataset results :**

Wiki-Vote Dataset :   
17-Clique is the Largest Clique found.

A total of ~460Thousand Maximal Cliques found.

Enron-Email Dataset :

20-Clique is the Largest Clique found.

A total of ~226Thousand Maximal Cliques were identified.

As-Skitter Dataset :

67-Clique is the Largest Clique.

Total 37Million Maximal Cliques are present.

**4. REAL-WORLD APPLICATION**

**4.1 Social Network Analysis:** Used to identify strongly connected communities or friend groups in social media platforms like Facebook, Twitter, and LinkedIn.

**4.2 Cybersecurity:** Helps in detecting coordinated attack groups in cyber networks by identifying dense clusters of malicious IPs or user accounts.

**4.3 Search Engines & Web Mining:** Helps in link analysis to detect communities or clusters of web pages with highly interconnected hyperlinks (e.g., Google's PageRank improvements).  
  
**4.4 Bioinformatics & Computational Biology:** Helps in Protein Interaction Networks by finding clusters of proteins that interact closely.

**4.5** **Computational Chemistry & Material Science:** Helps in Chemical Compound Analysis by finding molecular structures with similar bonding patterns.

**4.6 Linguistics & Natural Language Processing:** Helps in Synonym Clustering and Topic Detection by identifying groups of words frequently co-occurring together.  
  
**4.7 Telecommunications & Network Optimization:** Helps in Wireless Network Optimization by identifying groups of strongly connected nodes for efficient routing.

**5. CONCLUSION**

In this assignment we went through majorly three major algorithm to find the maximal cliques in graph

**Bron-Kerbosch** algorithm with pivoting proved to be efficient for medium-sized and sparse datasets, and pivoting helped to reduce the states. Basically, the pivoting idea is that rather than iterating at every vertex in the candidate set at every recursive step, we select a pivot vertex, **u**, and iterate over a smaller set. That smaller set is the candidate set minus the neighborhood of **u**. The idea behind this is that any clique that exists in the graph must contain vertex **u** or a non-neighbor of **u**. If a clique does not contain a non-neighbor of **u**, then **u** could be added to the clique to make it larger.

**Tomita’s** algorithm is an advanced improvement of the Bron-Kerbosch algorithm, which improves scalability and performance in dense subgraphs. Basically, in this, the selection of the pivot element is changed. It is selected such that it is a vertex with the most neighbors in the candidate set. This helps minimize the number of vertices in the candidate set that are not neighbors of the pivot, which in turn reduces the number of recursive calls and reduces the overall time complexity.

The **arboricity-based** algorithm performs very well on big, sparse graphs such as As-Skitter because it exploits the structure of the graph. It prioritizes lower-degree nodes and employs intelligent edge traversal to minimize unnecessary checks. This results in faster and less memory-intensive computation compared to other approaches to finding maximal cliques.

Working with actual datasets, we saw distinct trade-offs among accuracy, time complexity, and scalability across the various algorithms. This clearly indicated that the right algorithm entirely hinges on the type of dataset. Generally, the project highlighted how effective graph theory is in addressing real-world problems—from **social networks to bioinformatics**.

**6. REFERENCES:**

<https://arxiv.org/abs/1006.5440>

<https://www.sciencedirect.com/science/article/pii/S0304397506003586>

<https://www.cs.cornell.edu/courses/cs6241/2019sp/readings/Chiba-1985-arboricity>

<https://snap.stanford.edu/data/email-Enron.html>

<https://snap.stanford.edu/data/as-Skitter.html>

<https://snap.stanford.edu/data/wiki-Vote.html>