Group no 58
Id- f20221374@hyderbad.bits-pilani.ac.in
Name - Bharat Gupta

Contribution to the group
I am the only member of in group
I had alone done all things including deployment ,optimisation,report and webpage formation

Bron-Kerbosch Degeneracy Algorithm

Alright, let me explain it in a more natural way.

The **Bron–Kerbosch algorithm** is a way to find **all maximal cliques** in an **undirected graph**. A **clique** is just a group of nodes where every node is connected to every other node in that group. A **maximal clique** is one that you can't make bigger by adding more nodes.

Now, the **Bron–Kerbosch algorithm with degeneracy ordering** is an improved version that speeds things up, especially for sparse graphs.

What's degeneracy, and why does it help?

- The **degeneracy** of a graph is the highest number d such that, in any smaller part of the graph, there's at least one node with at most d connections.
- This helps because if we order the nodes **from least connected to most connected**, we can process the graph in a way that avoids unnecessary work.

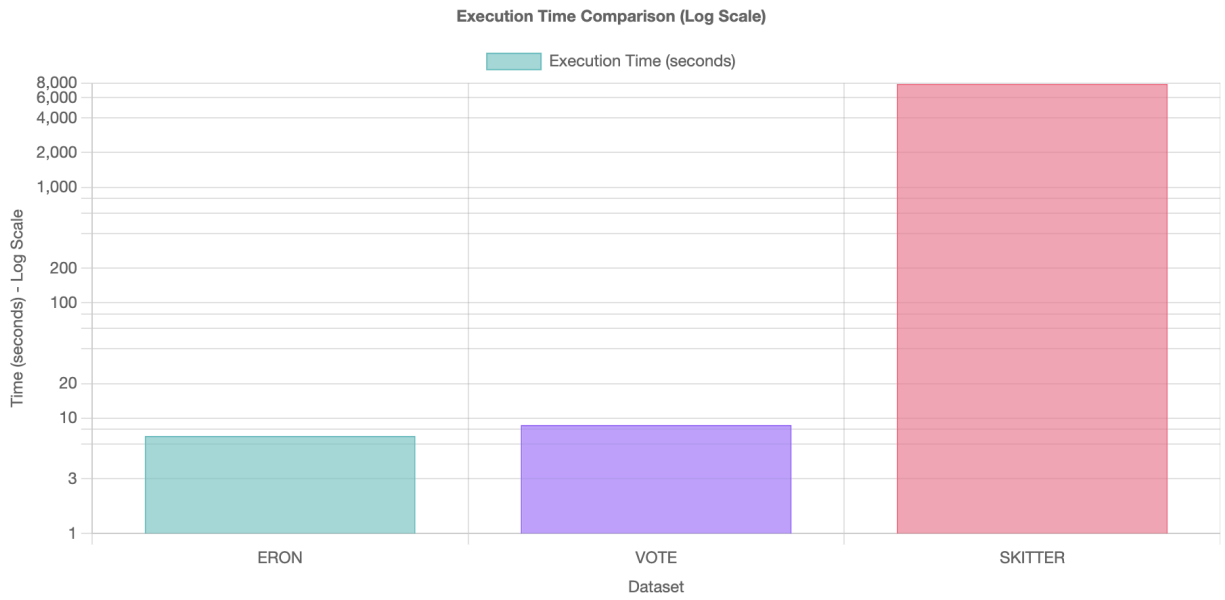
How does the algorithm work?

1. **Find the degeneracy ordering**
    - Take the node with the fewest edges, remove it, and remember the order.
    - Update the degrees of the remaining nodes.
    - Keep doing this until all nodes are removed.
    - This step is fast—takes O(n+m)) time (where n is the number of nodes, and mmm is the number of edges).
2. **Go through the nodes in reverse order**

- Start with the most connected node (since we ordered them from least to most).
- For each node v, look at its neighbors **that appear later in the ordering**.
- Run the **Bron–Kerbosch recursive function** on those neighbors to find maximal cliques.
- It also uses **pivoting**, which helps cut down the number of recursive calls.

Why is this faster?

- The original Bron–Kerbosch method can take as long as O(3^(n/3)) in the worst case.
- With degeneracy ordering, the runtime improves toO(d · 3^(d/3)) where d is the degeneracy (which is much smaller than n for sparse graphs).



Dataset	Execution Time (s)	Largest Clique Size	Total Maximal Cliques
ERON DATASET	7.02	20	226,859
Vote DATASET	8.76	17	459,003
SKITTER DATASET	7889.60	67	37,322,355

So, in simple terms: **we first order nodes cleverly, then process them efficiently, making the whole thing much faster!**

Tomita Algorithm explained explained

Tomita's Algorithm for Maximal Clique Enumeration

Tomita's algorithm is an optimized method for finding **all maximal cliques** in an **undirected graph**. It improves upon the **Bron–Kerbosch algorithm** by introducing **pivoting** and **degeneracy ordering**, making it significantly faster, especially for sparse graphs.

Key Optimizations:

1. **Pivot Selection:** Instead of blindly exploring all possibilities, it selects a **pivot vertex** that helps minimize the number of recursive calls. The pivot is chosen to cover as many candidates as possible, reducing unnecessary searches.

2. **Degeneracy Ordering:** The graph is processed in an order where each vertex has at most a small number of later neighbors. This ensures that the search space remains small at every step.

How It Works:

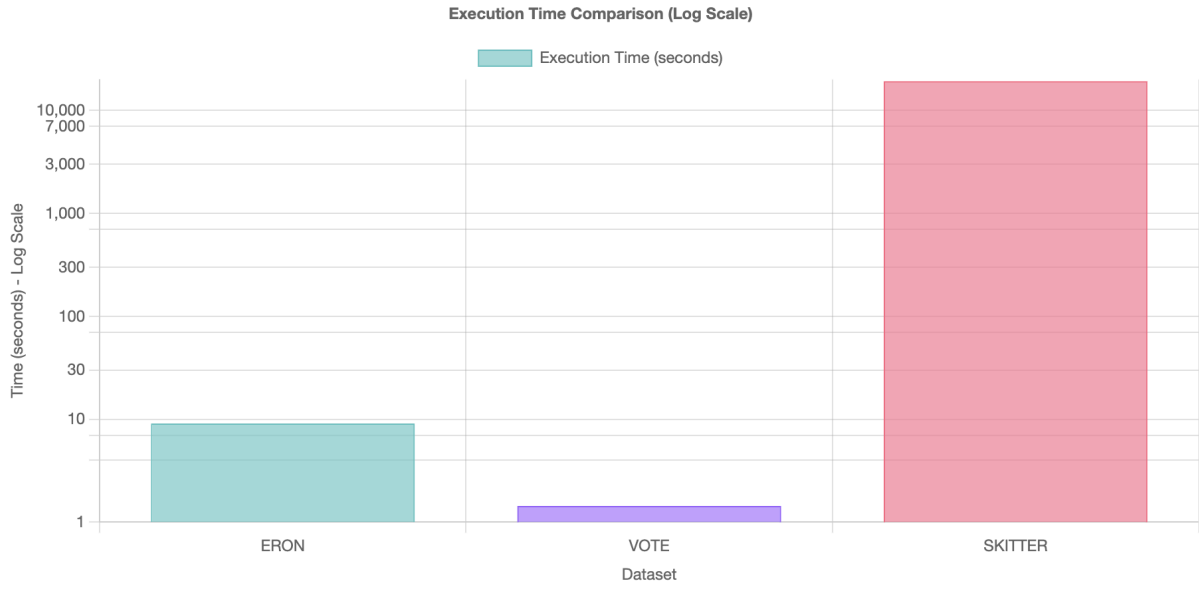
- The algorithm starts with an empty clique and explores **only the necessary nodes** based on pivoting.

- It **expands** cliques recursively, ensuring no clique is missed while avoiding redundant checks.

- By processing nodes in **reverse degeneracy order**, it reduces the number of branches to explore, making it more efficient than standard methods.

This approach significantly improves performance, especially for large real-world graphs, by avoiding unnecessary computations and focusing on the most promising candidates.

The chart below compares the execution times across all three datasets:



Dataset	Execution Time (s)	Largest Clique Size	Total Maximal Cliques
ERON DATASET	9.09	20	226,859
VOTE DATASET	1.44	17	459,003
SKITTER DATASET	19,226.05	67	37,322,355

Chiba Algorithm

Chiba and Nishizeki’s Algorithm for Maximal Clique Enumeration

Chiba and Nishizeki’s algorithm is a highly efficient method for finding all maximal cliques in a graph, particularly optimized for sparse graphs. Instead of using brute-force approaches, it leverages the "k-core decomposition" and a recursive edge-based search to minimize unnecessary computations.

Key Ideas:

1. K-Core Decomposition: The algorithm first orders vertices based on their degrees and processes them in increasing order. This ensures that each vertex is examined only when it has a low degree, reducing complexity.

2. Edge-Based Search: Rather than examining all subsets of vertices, it focuses on edges and extends them into larger cliques by adding only the vertices that are directly connected to all members.

3. Efficient Pruning: Since vertices are processed in increasing order of degree, the algorithm can quickly discard unpromising candidates, making it much faster than naive approaches.

How It Works:

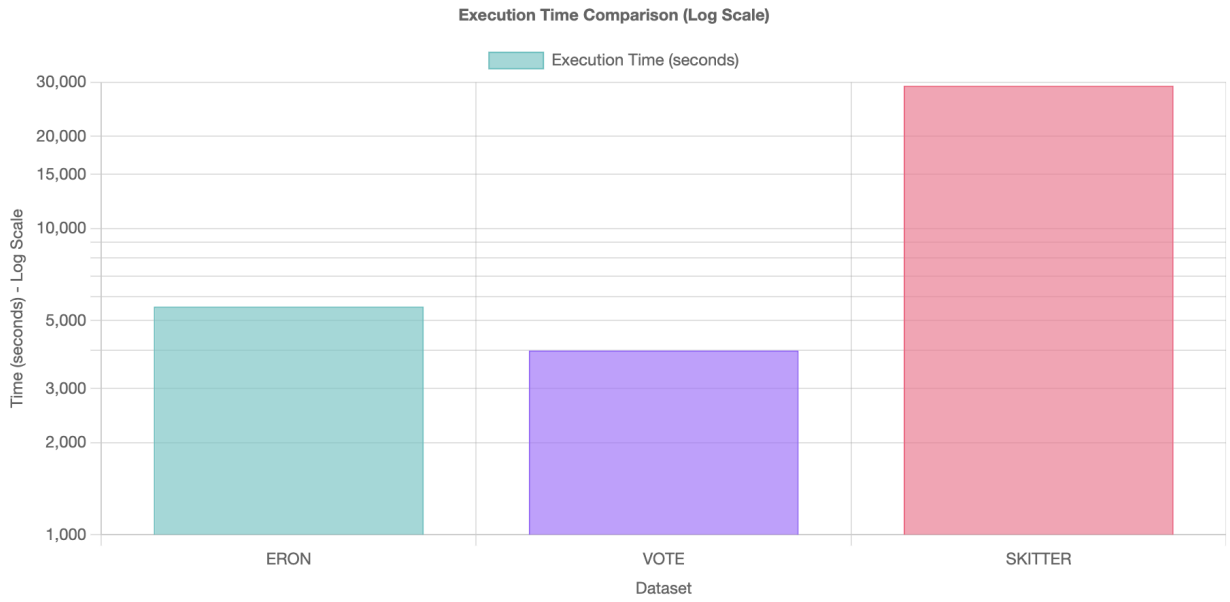
- The algorithm begins by sorting vertices based on their degrees and iteratively removes the smallest-degree vertex, ensuring that each vertex is handled at the right time.

- For each vertex, it searches for maximal cliques by extending edges into larger cliques while ensuring all included vertices remain connected.

- Since it processes only low-degree vertices first, the number of checks reduces significantly, leading to an efficient enumeration of maximal cliques.

This method is particularly useful for graphs that are sparse (i.e., those with relatively few edges compared to the number of vertices), as it avoids the combinatorial explosion that other algorithms might face.

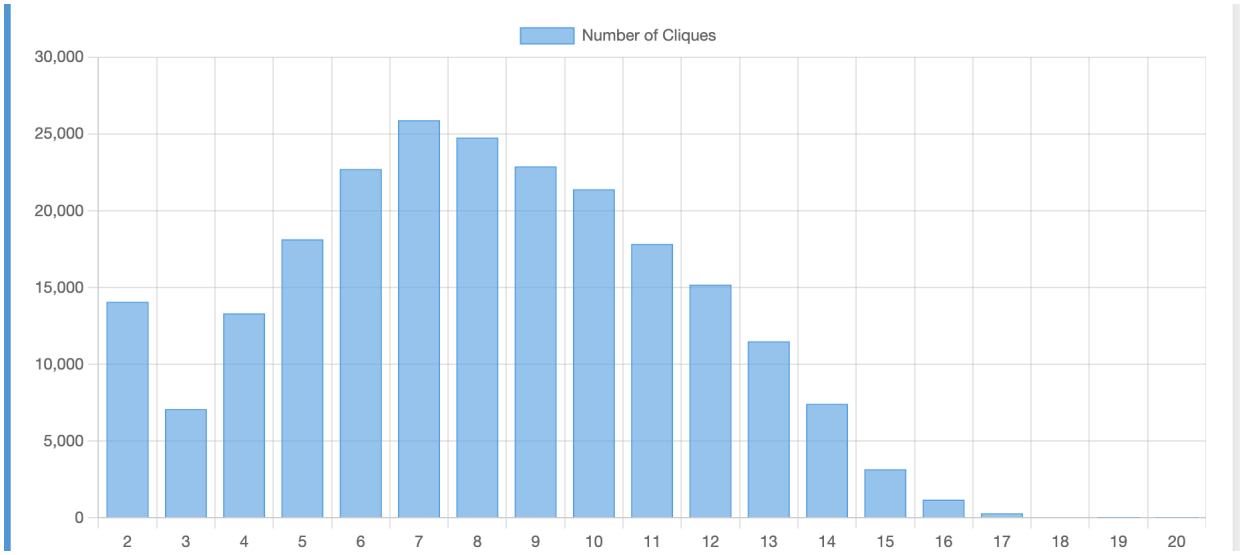
The chart below compares the execution times across all three datasets:



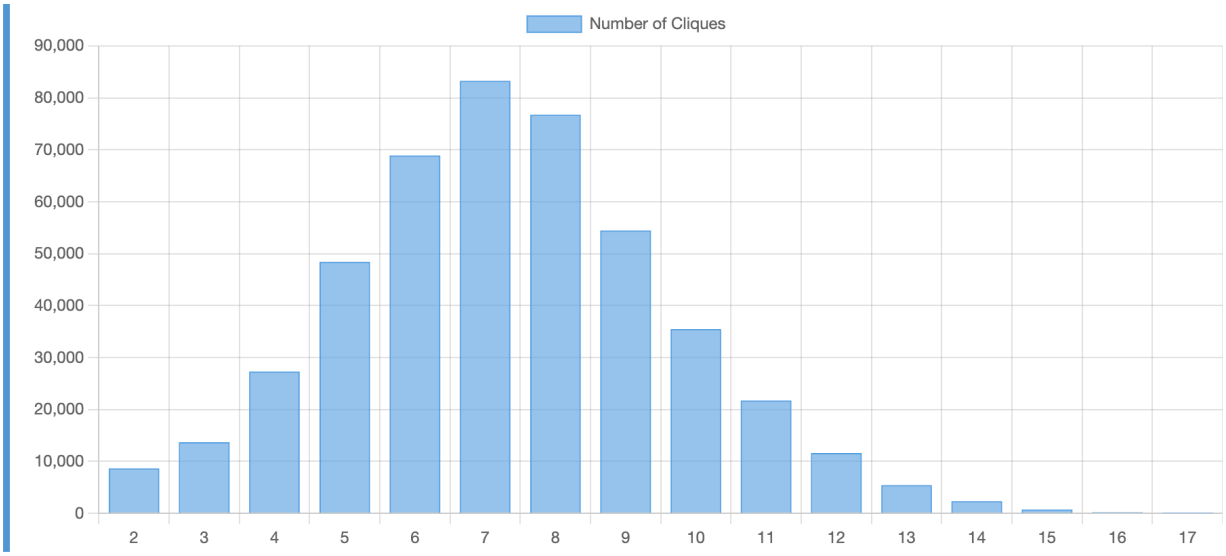
Dataset	Execution Time (s)	Largest Clique Size	Total Maximal Cliques
ERON DATASET	5563.90	20	226,859
VOTE DATASET	3998.80	17	459,003
SKITTER DATASET	29,256.005	67	37,322,355

Histogram of all datasets

Enron dataset

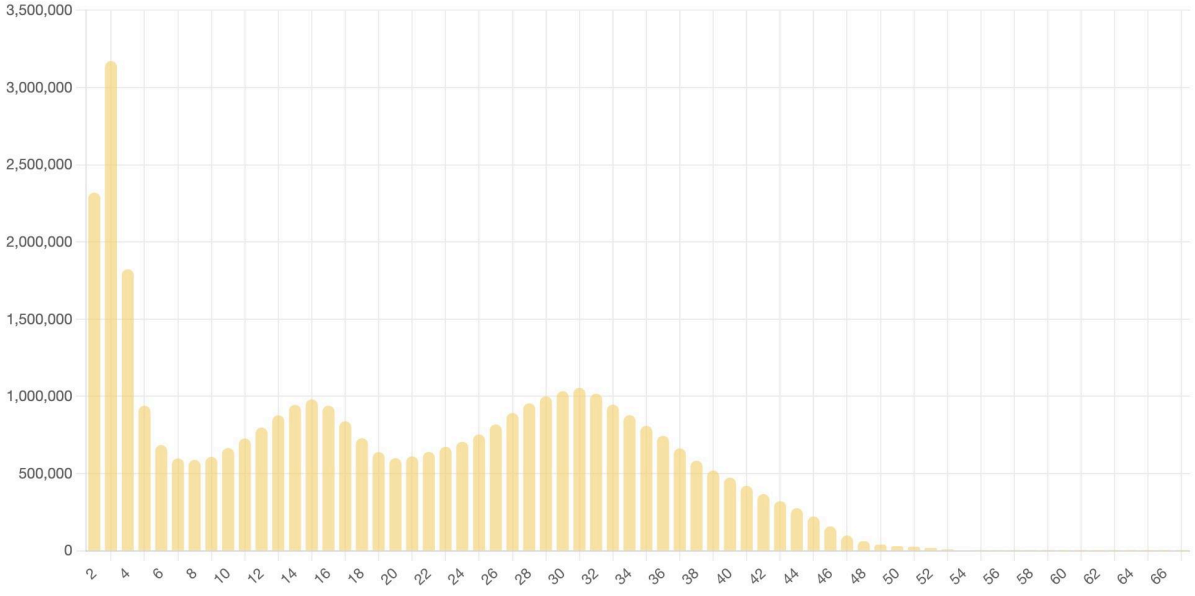


Vote dataset



SKITTER .TXT

SKITTER.TXT


3,500,000
3,000,000
2,500,000
2,000,000
1,500,000
1,000,000
500,000
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40
42
44
46
48
50
52
54
56
58
60
62
64
66