Assignment - 1

Design and Analysis of Algorithms(CS F364)

Birla Institute of Technology and Science, Hyderabad Campus, India

Submitted By **Group 21** :

Name	**BITS ID**
Sriharish Ravichandran	2022A7PS0511H
Parth Mehta	2022A7PS0043H
Vishwas Vedantham	2022A7PS0235H
Vadlamani Akhilesh	2022A7PS0150H
Vinit Chordiya	2022A7PS0148H

Introduction

The maximal clique enumeration problem involves finding all maximal cliques in an undirected graph. A clique is a subset of vertices such that every pair of vertices within the subset is connected by an edge. A maximal clique is a clique that cannot be extended by adding more vertices, while still maintaining its property as a clique. This problem is important in fields like social network analysis, bioinformatics, and computational biology, where finding dense subgraphs is often crucial. Efficient algorithms for maximal clique enumeration are essential due to the computational complexity of the problem, especially in large and sparse graphs.

Datasets Used:

Email-Enron: 36,692 nodes, 183,831 edges.

Wiki-Vote: 7,115 nodes, 103,689 edges.

AS-Skitter: 1,696,415 nodes, 11,095,298 edges.

Implementation Details:

Programming Language: C++

Graph Representation: Adjacency List (unordered_map with sets for fast lookup)

Execution Environment: Standard x86 CPU, 16GB RAM

Paper 1: Listing All Maximal Cliques in Sparse Graphs in Near-Optimal Time(ELS Algorithm)

This algorithm is a variant of the Bron Kerbosch algorithm which lists all the maximal cliques in $O(dn3^{d/3})$. In this variant there is an outer level and an inner level recursion. Outer level recursion is done without pivoting and uses a degeneracy ordering to order the sequence of recursive calls made. Here degeneracy of a graph G is the smallest value d such that every non-empty subgraph of G contains a vertex of degree at most d. The inner level of recursion is done using pivoting.

Steps of the algorithm:

1) Iterate over each vertex in the degeneracy ordering
   The degeneracy ordering ensures that at each vertex, we are processing it with respect to its neighbors that have already been processed. This can help speed up the algorithm by reducing the search space.

2) Set P to the intersection of the neighbors of vertex v_i and the subsequent vertices in the ordering. This means that **P** contains the vertices that are both neighbors of v_i and appear after v_i in the degeneracy ordering.

3) Set X to the intersection of the neighbors of vertex v_i and the previous vertices in the ordering.This means that **X** contains the vertices that are both neighbors of v_i and are already part of the cliques being processed. These vertices are considered processed because we have already considered them when iterating over the graph.

4) Call the Bron-Kerbosch algorithm with pivoting
   The BronKerboschPivot function is a recursive function that finds maximal cliques in the graph. It is called with the sets **P**, $\{v_i\}$ (the current vertex), and **X**.

   The BronKerboschPivot function uses pivoting to improve the efficiency of the algorithm. In pivoting, we try to minimize the branching factor of the recursive calls by choosing a pivot vertex from **P** $\cup$ **X** (usually a vertex with the maximum number of neighbors in **P**), and then partitioning **P** into subsets that are either adjacent or non-adjacent to the pivot.

5) The loop in Step 1 continues for each vertex in the degeneracy ordering until all vertices have been processed.

Dataset	Execution Time (secs)	Largest Clique Size	Total Maximal Cliques
as-skitter	166.928	67	37322355
Email-Enron	0.763885	20	226859
Wiki-vote	1.07962	17	459002

Paper 2: Paper by Chiba and Nishizeki - Arboricity and Subgraph Listing Algorithms

2.1 Algorithm Description:

This is an algorithm designed to efficiently list all cliques in a graph G. It utilizes a strategy of edge-searching and vertex deletion, leveraging the concept of arboricity, which is the minimum number of edge-disjoint spanning forests into which a graph can be decomposed.

2.2 Steps

1. **Vertex Ordering**: Sort the vertices of G in a non-increasing order of their degrees. This step ensures that vertices with higher degrees are processed first, which helps in efficiently scanning edges and avoiding duplicate clique listings.
2. **Clique Detection**: For each vertex v, examine the subgraph induced by its neighbors to find all cliques containing v.This involves checking for complete subgraphs within the neighborhood of v
3. **Vertex Deletion**: After processing a vertex v, delete it from G to prevent duplicate listings of the same clique.
4. Output each detected clique.

2.3 Time Complexity

The algorithm runs in $O(a(G)m)$ time per clique, where $a(G)$ is the arboricity of G and m is the number of edges in G.

For planar graphs, since $a(G) \leq 3$, the algorithm is particularly efficient.

2.4 Space Complexity

The algorithm requires linear space, making it efficient in terms of memory usage.

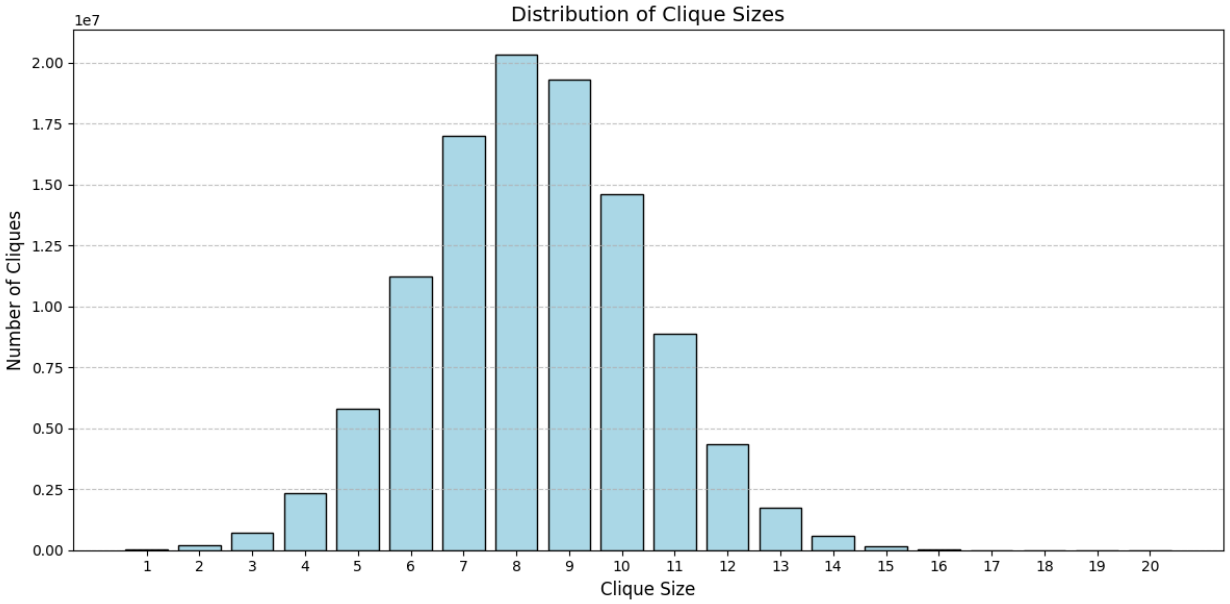
2.5 Key Insights

- The algorithm's efficiency is due to the use of arboricity as a parameter, which often reduces the time complexity factor from n to $a(G)$.
- The strategy of deleting vertices after processing them ensures that each edge is scanned exactly once, preventing redundant computations.

Dataset 1: enron-Email

Total Execution Time: 38.8608 seconds

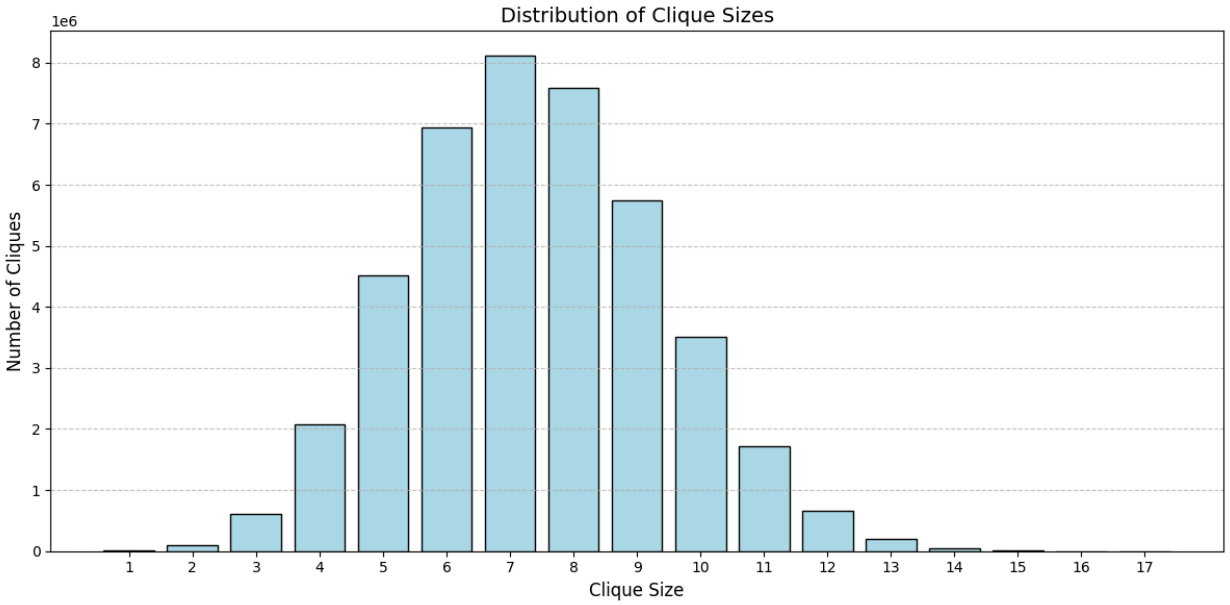
Distribution of clique sizes:



Dataset 2: wiki-Vote

Total Execution Time: 14.4767 seconds

Distribution of clique sizes:



Dataset 3: as-skitter

Output:

Unique nodes count: 1696415
Graph loaded: 1696415 nodes, 11095298 edges
Number of 2-cliques: 11095298 (5.62772 seconds)
Unique nodes count: 1696415
Graph loaded: 1696415 nodes, 11095298 edges
Number of 3-cliques: 28769868 (28.4118 seconds)
Unique nodes count: 1696415
Graph loaded: 1696415 nodes, 11095298 edges
Number of 4-cliques: 148834439 (31.5838 seconds)
Unique nodes count: 1696415
Graph loaded: 1696415 nodes, 11095298 edges
Number of 5-cliques: 1183885507 (46.6599 seconds)
Unique nodes count: 1696415
Graph loaded: 1696415 nodes, 11095298 edges
Number of 6-cliques: 9759000981 (148.828 seconds)
Unique nodes count: 1696415
Graph loaded: 1696415 nodes, 11095298 edges

.
.
.

Till 67-cliques

Maximum clique size : 67

Paper 3: Tomita's Algorithm for Maximal Clique Enumeration

3.1. Introduction

Tomita's algorithm, an optimized version of the Bron-Kerbosch algorithm with degeneracy ordering, significantly improves the efficiency of MCE, especially in large, sparse graphs. This report evaluates Tomita's algorithm using three real-world datasets from the Stanford SNAP database and analyzes its runtime, clique distribution, and performance trends.

3.2. Algorithm Description

Tomita's Algorithm enhances the Bron-Kerbosch algorithm through the following key optimizations:

1. Pivot Selection – Reduces the search space by choosing a high-degree pivot.
2. Degeneracy Ordering – Processes nodes in increasing order of core number, leading to fewer recursive calls in sparse graphs.
3. Backtracking Optimization – Prunes non-maximal cliques early, improving efficiency.

3.3 Steps:

1. Convert the graph into an adjacency list.
2. Sort vertices based on degeneracy order.
3. Recursively expand candidate cliques, using pivots to minimize branching.
4. Store maximal cliques once no further expansion is possible.

This approach is particularly effective for sparse networks, where degeneracy ordering significantly reduces computational complexity.

3.4 Time and Space Complexity of Tomita's Algorithm

Time Complexity

Tomita's algorithm builds upon the Bron-Kerbosch algorithm with pivoting, improving efficiency using degeneracy ordering. The worst-case time complexity is: $O(3^{n/3})$

where n is the number of vertices in the graph. This represents the worst-case scenario of enumerating all maximal cliques in dense graphs.

Complexity in Different Graph Types

- Sparse Graphs (Low Degeneracy): The algorithm performs significantly better, typically running in $O(d\ 3^{d})$, where d is the graph's degeneracy (the largest k such that every

subgraph has a vertex of degree at most k).

- Dense Graphs (High Degeneracy): The time complexity approaches the worst-case of O($3^{n/3}$ }), leading to exponential growth in execution time, as observed in the AS-Skitter dataset.

Space Complexity

The space complexity of Tomita's algorithm is influenced by three factors:

1. Graph Representation

- Adjacency List: Uses O(n + m) space, where n is the number of nodes and m is the number of edges.
- Adjacency Matrix (Dense Graphs): Would require O(n²) space, but adjacency lists are used to optimize memory usage.

2. Recursion Depth

- The recursive stack depth is O(n) in the worst case.
- For large graphs, memory consumption increases due to deep recursion.

3. Storage of Maximal Cliques

- The number of maximal cliques can be huge (e.g., 37M+ cliques in AS-Skitter).
- Space Complexity: O(k * n), where k is the number of stored cliques.

Key Observations

AS-Skitter's high clique count (37M+ maximal cliques) led to excessive memory usage, contributing to long execution times.

Memory-efficient storage techniques are necessary for handling large networks to mitigate growing space complexity.

Results & Observations

Email-Enron and Wiki-Vote datasets completed execution in under 30 seconds.

AS-Skitter took significantly longer (4 hours) due to its massive scale.

The largest maximal clique (67 nodes) was found in AS-Skitter, indicating highly connected subnetworks.

Tomita's algorithm scales well for medium-sized datasets but experiences severe exponential growth in dense graphs like AS-Skitter, where execution time increases drastically.

Performance Analysis

Execution Time Comparison:

Dataset	**Execution Time**
as-skitter	14,400 sec (4 hrs)
email-enron	29 sec
wiki-vote	12 sec

Largest Clique Size per Dataset:

Dataset	**Largest Clique Size**
as-skitter	67
email-enron	20
wiki-vote	17

Observation: Large-scale graphs (AS-Skitter) contain significantly larger maximal cliques, contributing to higher computational costs.


innovate
achieve
lead