Report on Maximal Clique Analysis

“The worst-case time complexity for generating all maximal cliques and computational experiments” paper implementation results:

Largest Size of the Clique in each dataset:

- wiki-Vote: 17
- email-Enron: 20
- as-Skitter: 67

Total Number of Maximal Cliques in each dataset:

- wiki-Vote: 459,002
- email-Enron: 226,859
- as-Skitter: 37,322,355

Execution time on each dataset:

- wiki-Vote: 2.572 seconds
- email-Enron: 2.649 seconds
- as-Skitter: 6298.71 seconds

“Listing All Maximal Cliques in Sparse Graphs in Near-optimal Time” paper implementation results:

Largest Size of the Clique in Each Dataset:

- **email-Enron**: Largest maximal clique size is 20
- **wiki-Vote**: Largest maximal clique size is 17
- **as-Skitter**: Largest maximal clique size is 67

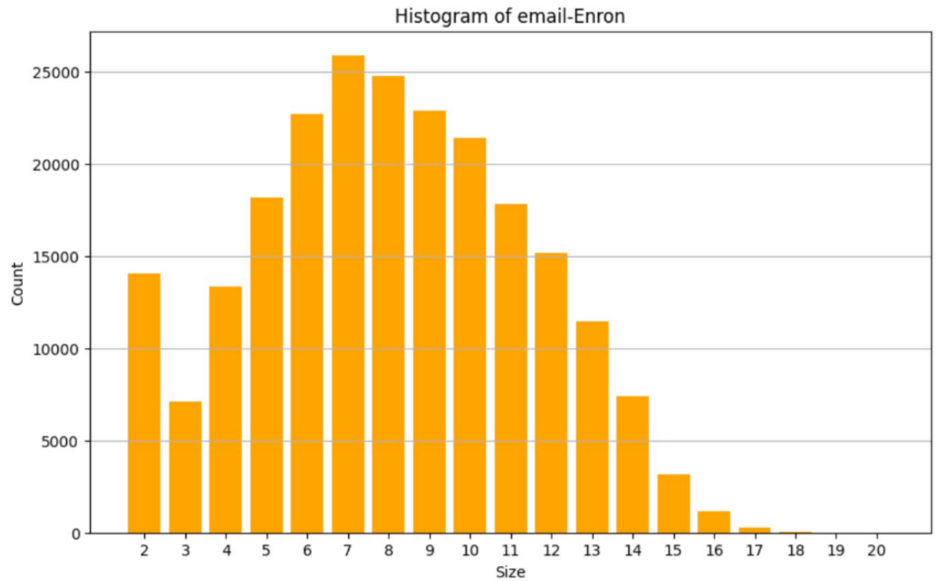
Total Number of Maximal Cliques in Each Dataset:

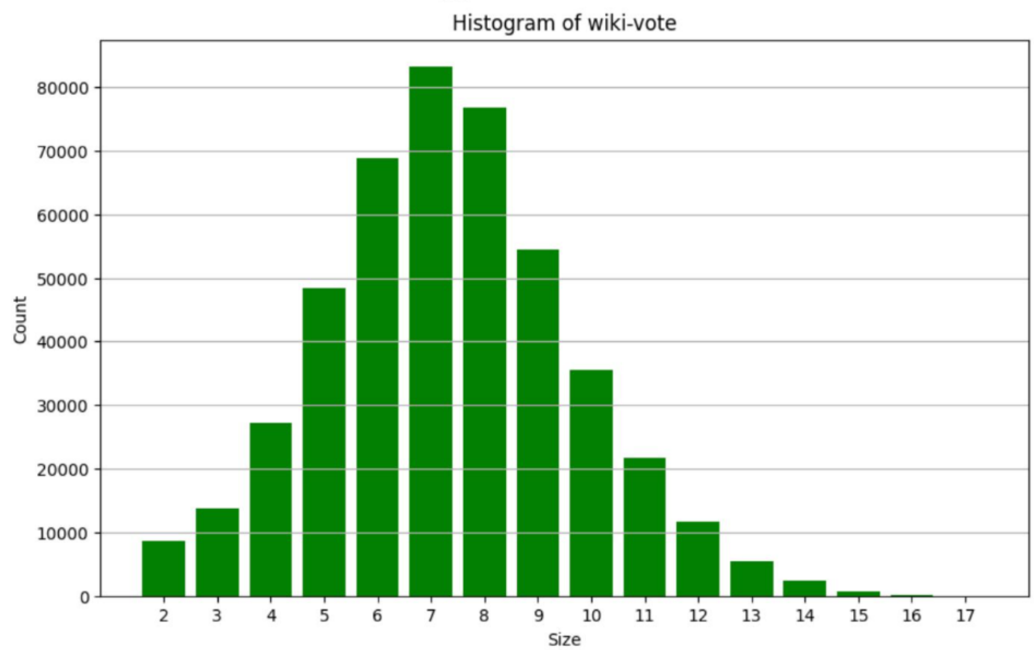
- **email-Enron**: 226,859 maximal cliques
- **wiki-Vote**: 459,002 maximal cliques
- **as-Skitter**: 37,322,355 maximal cliques

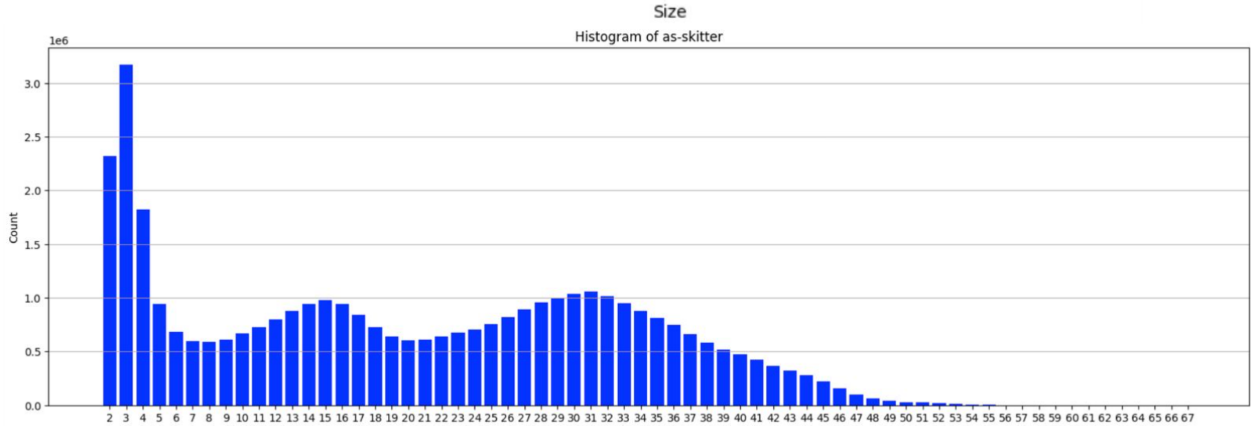
Execution Time on Each Dataset:

- **email-Enron**: 3.197 seconds

- **wiki-Vote**: 2.953 seconds
- **as-Skitter**: 1567.462 seconds (approx 26.1 minutes)







Insights:

Execution Time Comparison

- The execution times for email-Enron(3.197s) and wiki-Vote (2.953s) are very close, meaning the dataset size and structure didn't significantly affect the runtime .

- Whereas as-Skitter(1567.462s) took longer, showing that it is on a higher scale in terms of complexity.This suggests that its graph structure and number of maximal cliques are significantly larger.

Maximal Clique Count Comparison

- wiki-Vote(459,002 cliques) has twice as many maximal cliques as email-Enron(226,859 cliques), yet their execution times are almost identical. This suggests that the increase in clique count did not impact runtime here.

- as-Skitter(37.3M cliques) has roughly 80x more cliques than the other two, and the execution time reflects this extreme growth clearly. **This shows that Clique enumeration exponentially increases with the size of the dataset.**

Largest Maximal Clique Size Comparison

- largest clique:
  - as-Skitter(67)
  - email-Enron(20)
  - wiki-Vote (17)

- Despite wiki-Vote and email-Enron having similar execution times, the largest clique in wiki-Vote is slightly smaller,which indicates that larger maximal cliques contribute more to computation.

Distribution of Clique Sizes

- In both email-Enron and wiki-Vote, the majority of cliques are in the size range of 2 to 10, meaning most of the graph has small cliques.

- wiki-Vote has more cliques of size 6-10, which indicates a denser network in certain parts of the graph.

- Since as-Skitter has cliques up to size 67, it has a more interconnected structure compared to the other datasets.

About the Algorithm:

Bron-Kerbosch Algorithm

The Bron-Kerbosch algorithm is a recursive backtracking algorithm for finding all maximal cliques in an undirected graph. The pivoting strategy improves efficiency by reducing the search space.

Key Concepts:

1. **Clique:**
   - A clique is a subset of vertices such that every pair of vertices in the subset is connected by an edge.
2. **Maximal Clique:**
   - A clique is maximal if it cannot be extended by including any other vertex.
3. **Pivoting:**
   - A pivot vertex is chosen to minimize the number of recursive calls by focusing only on vertices not connected to the pivot.

Steps in the Algorithm:

1. **Initialization:**

   - Start with three sets:
      - R: Vertices currently forming a clique.
      - P: Vertices that can still be added to the clique.
      - X: Vertices that have already been processed and cannot be added.
2. **Recursive Backtracking:**
   - If both P and X are empty, then R forms a maximal clique, which is added to the result.
   - Otherwise:
      - Choose a pivot vertex from P**UX**.
      - Restrict exploration to vertices in P that are not neighbors of the pivot.
      - For each such vertex:
         - Add it to R.
         - Update P and X based on its neighbors.
         - Recursively call the algorithm with updated sets.
         - Remove the vertex from R, move it from P to X.
3. **Degeneracy Ordering:**
   - To optimize performance, vertices are processed in degeneracy order (low-degree vertices first). This reduces recursive calls by minimizing candidate vertices (P).
4. **Clique Size Counting:**
   - Each maximal clique's size is recorded in a map (cliqueSizeCount) for histogram generation.

Paper: Arboricity and Subgraph Listing Algorithms

explanation of the algo:-

Preprocessing Step (Graph Ordering)

Step 1: Number the vertices in non-decreasing order of their degree: $d(1) \le d(2) \le \ldots \le d(n)$. This ensures that low-degree vertices are processed first, which helps in early pruning of search space.

Step 2: Initialize two auxiliary arrays S[y] and T[y] for all vertices:
S[y] → Counts the number of neighbors of y in C-N(i).
T[y] → Counts the number of neighbors of y in C∩N(i).

Step 3: Initialize an empty clique C and call UPDATE(2, C) to start recursion.

****Recursive Clique Expansion (UPDATE Function):-**

The UPDATE(i, C) procedure recursively constructs a clique C by adding vertex i and checking whether it is a valid maximal clique.

Base Case: When all vertices are processed
If i=n+1, print the clique C, since no additional vertices can be added.
Backtrack to the last recursive call.

Step 1: Guarantee Clique Validity

If C∩N(i)≠Ø(there exists at least one neighbor in CC, then
Call UPDATE(i + 1, C) (keep exploring without including i).

Step 2: Calculate Neighboring Counts S[y] and T[y]
Calculate T[y] for all the vertices y∉C (y neighbors that are in C∩N(i).

Calculate S[y] for all the vertices $y \notin C$ (y neighbors that are in C-N(i).
This information aids in testing maximality and lexicographic order.

Step 3: : Maximality Test
If a vertex y in N(i)-C has all its neighbors already in $C \cap N(i)$, discard C.
This ensures that only maximal cliques are considered.

Step 4: Lexicographic Order Check

Sort vertices in C-N(i) in ascending order j1,j2,…,jp.
Check that inserting i preserves lexicographic order.
If C is not lexicographically valid, reject it.

Step 5: Restore S[y] and T[y]
Clear values after testing to ready for the next iteration.

Step 6: If Valid, Expand the Clique
Save C-N(i) as SAVE (vertices which won't be taken).
Update C to take i and recurse with UPDATE(i + 1, C).
After recursion, restore C to consider other possibilities.

Time Analysis of Chiba And Nishizeki's "Arboricity and subgraph listing":

Arboricity

Arboricity is a measure of graph sparsity(or density), defined as the minimum number of forests into which the edges of a graph can be partitioned. For a graph G with n vertices and m edges, the arboricity a(G) is bounded by:

a(G)≤⌈root(2m+n)⌉.

Chiba and Nishizeki's algorithm provides efficient subgraph listing for:

1. Triangles: O(mα) time
2. 4-cycles: O(mα + t) time, where t is the number of 4-cycles
3. k-cliques: O(mα^(k-2)) time, for k ≥ 4

Here, m is the number of edges, and α is the graph's arboricity.

Efficiency: The algorithms are particularly efficient for graphs with low arboricity. For planar graphs (α ≤ 3), triangle and 4-cycle listing become linear time operations

Execution Time on Each Dataset:

- **email-Enron**: 1256.32 seconds
- **wiki-Vote**: 132.05 seconds
- **as-Skitter**: NA(too long to run)

CONCLUSION :- In given datasets,email-enron has low arboricity compared to wiki-vote , thus even though the number of edges is low in wiki-vote it's taking a lot of time compared to email-enron. where as-skitter it's both very very large and has high arboricity thus taking even more time which is practically not feasible.

Time analysis of 3 Algorithms with the 3 given Datasets

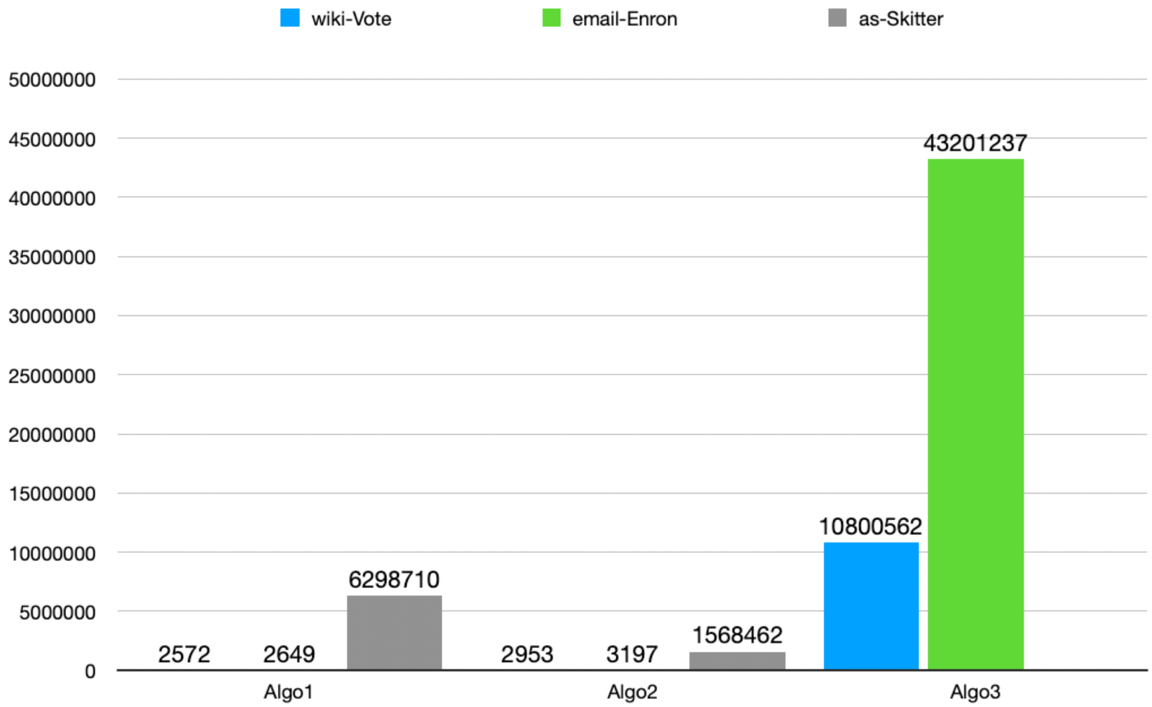


Table 1

	Algo1	Algo2	Algo3
wiki-Vote	2572	2953	10800562
email-Enron	2649	3197	43201237
as-Skitter	6298710	1568462	