DAA Assignment 1 - Report
Group - 32

Algorithm Descriptions

Paper 1 - Tomita

This algorithm is designed to find all maximal cliques in an undirected graph G = (V, E). The main procedure, CLIQUES, initializes a global variable Q to represent a clique and calls the recursive procedure EXPAND with two sets: SUBG = V (the set of all vertices in the graph) and CAND = V (the set of candidate vertices that can form cliques). In EXPAND, if SUBG is empty, it prints Q, indicating that Q is a maximal clique. Otherwise, it selects a pivot vertex u from SUBG that maximizes the intersection of CAND and the neighbors of u. It iterates over vertices in CAND - N(u), adding each vertex q to Q, updating SUBG, CAND, and recursively calling EXPAND to grow the clique. After exploring all possibilities for a vertex, it backtracks by removing q from Q. This ensures all maximal cliques are found through systematic exploration and backtracking.

Paper 2 - ELS

The procedure BronkerboschDegeneracy(V,E) is a variant of the Bron-Kerbosch algorithm designed to efficiently list all maximal cliques in a graph by leveraging its degeneracy. It starts by computing a degeneracy ordering of the vertices in the graph, which helps in organizing the recursive calls. The algorithm then iteratively applies the Bron-Kerbosch method with pivoting at inner levels of recursion. This involves selecting a pivot vertex that maximizes the intersection of candidate vertices and their neighbors, which reduces the number of recursive calls. By using the degeneracy ordering for the outer recursion and pivoting for inner recursion, the algorithm achieves a time complexity of O(d.n.3^(d/3)), making it efficient for sparse graphs with low degeneracy.

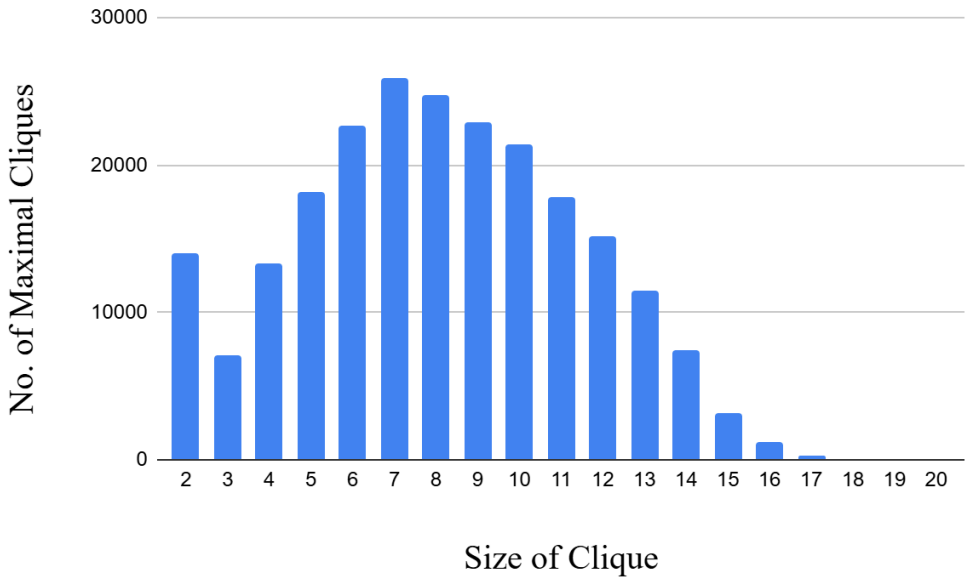
Paper 3 - Chiba

The procedure CLIQUE is designed to list all cliques in a graph efficiently. It starts by numbering the vertices in nondecreasing order of their degrees. The algorithm then recursively generates cliques by adding or removing vertices from the current clique. It uses two key tests: a "maximality test" to ensure the new clique is maximal, and a "lexico.test" to avoid duplicate cliques by checking if the current clique is lexicographically larger. The algorithm iterates through each vertex, updating the clique based on these tests, and prints out new cliques when conditions are met. This approach ensures that all cliques are found while minimizing computational time, achieving an efficiency of O(a(G).m) time per clique, where a(G) is the arboricity of the graph and m is the number of edges.

Observations

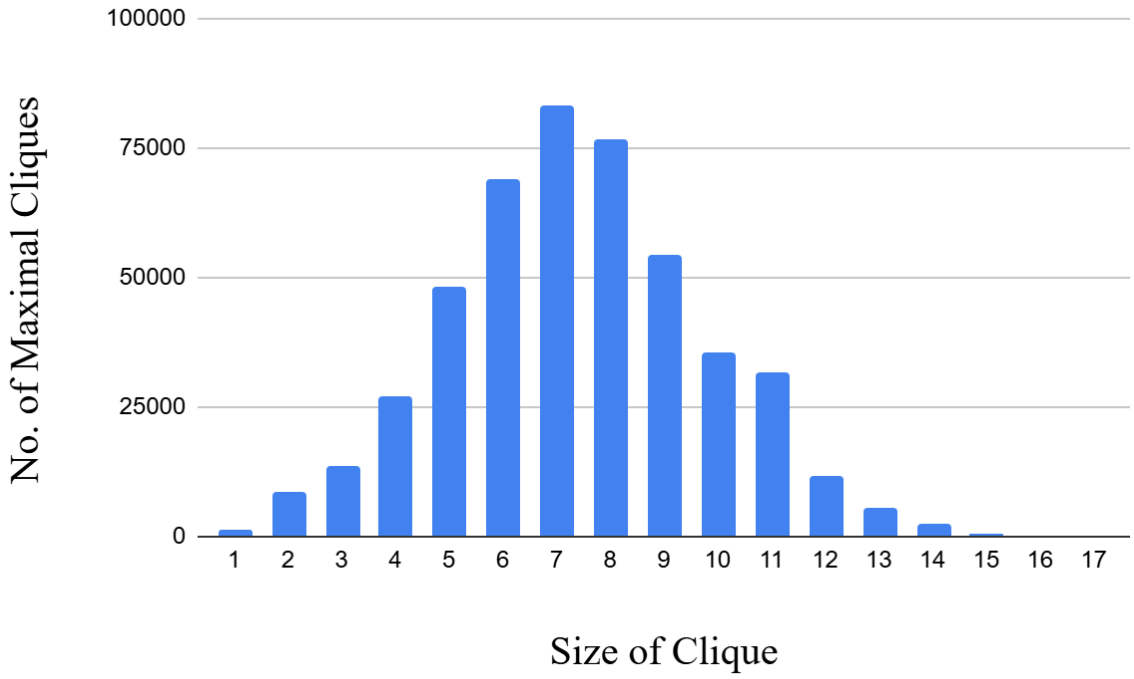
Dataset 1 - Enron

1. Largest Clique: Size = 20
   {2572 140 314 575 292 416 1185 1330 593 175 233 241 299 383 586 225 255 526 592 1320}
2. Total number of maximal cliques = 226859
3. Distribution of cliques by size (Histogram):



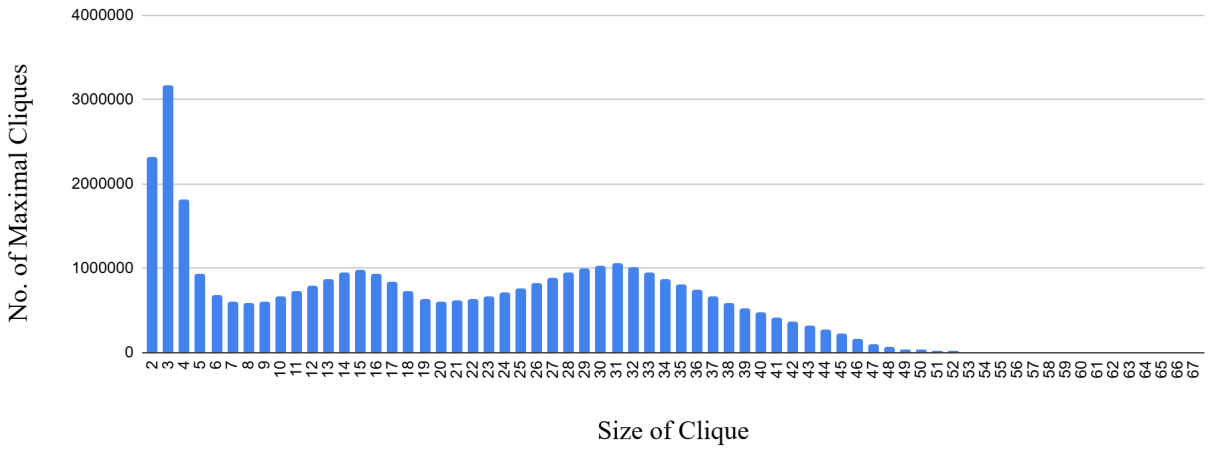
Dataset 2 - WikiVote

1. Largest Clique: Size = 17
   {2565 825 1549 3028 2297 1166 2674 2688 2485 2625 2654 2686 2660 3026 3258 2713 2700}
2. Total number of maximal cliques = 459002
3. Distribution of cliques by size (Histogram):



Dataset 3 - Skitter

1. Largest Clique: Size = 67
2. Total number of maximal cliques = 37322355
3. Distribution of cliques by size (Histogram):



Execution Time:

Paper 1 - Tomita

1. Enron: 7.257 seconds
2. WikiVote: 4.844 seconds
3. Skitter: 47800.285 seconds

Paper 2 - ELS

1. Enron: 0.974 seconds
2. WikiVote: 1.389 seconds
3. Skitter: 212.470 seconds

Paper 3 - Chiba

1. Enron: 0.644 seconds
2. WikiVote: 0.881 seconds
3. Skitter: 300.255 seconds