CS F364

Design and Analysis of Algorithms

ASSIGNMENT - 1

|  |  |
| --- | --- |
| Name | ID |
| Hrishikesh MK | 2022A7PS1538H |
| Satya K | 2021B3A70723H |
| Ashish C | 2021B3A73029H |
| Harinandan Arun | 2021B5A72396H |
| Yuvraj Chauhan | 2022A7PS0135H |

****

**Tomita’s Algorithm**

**Set of symbols used here**:

***Q*** – a set of vertices storing the subgraph in expand function (which is a subset of a maximal clique)

***V*** – the set of vertices in the overall graph

– the intersection set of vertices which **are adjacent to all nodes** in Q in the graph

***CAND*** – the set of candidates for e

***Γ(u)*** – the set of adjacent vertices of a vertex u

– the set of vertices after removing Γ(u) from CAND

***MAXCLIQUES*** – the set of maximal cliques

***MaxCliqueSize*** – the size of largest maximal clique observed until now

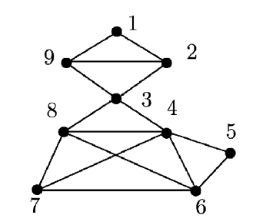
Tomita’s Algorithm for finding maximal cliques uses a depth first search (backtracking) method to generate all maximal cliques for an undirected graph. It makes use of a Global set Q, initially empty, which it expands using the expand function. Set Q stores subgraph constituting the cliques at each instance of expand function. Size of MAXCLIQUES at the end of the algorithm gives the number of cliques.  
  
The procedure of algorithm is fairly straightforward.

* The function Expand takes in two parameters SUBG and CAND both of which are initially set to V.
* If the SUBG = ∅, then we have found the maximal clique since there are no nodes who share an edge with all other nodes in Q (current clique.
  + Here we update the clique count
  + Print clique to notify user
  + Add Q to MAXCLIQUES
  + Also size of Q is compared with MaxCliqueSize and MaxCliqueSize is updated
* If not, then we find a vertex u in SUBG such that it maximizes |CAND ∩ Γ(u)| (maximize function is used for the same which utilizes binary search and linear time search to find the intersection).
  + is defined and while ≠ ∅, a vertex q is picked from . This vertex is a node in the search tree. We then add this vertex to our current clique Q.
  + We define = SUBG ∩ Γ(q) and = CAND ∩ Γ(q).
  + We then apply Expand function recursively with and as the parameters.
  + CAND and are updated by removing q
  + Q is updated by removing q (new clique) and print back to notify the user that we are moving to the previous state of vertices in Q to find a new clique.

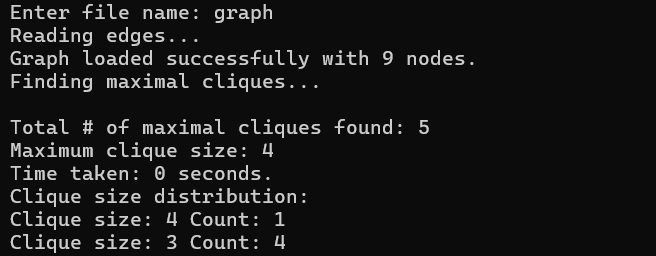
**Observations and Findings**

On applying the algorithm on multiple datasets, here is the output that was observed:

Dataset 1:

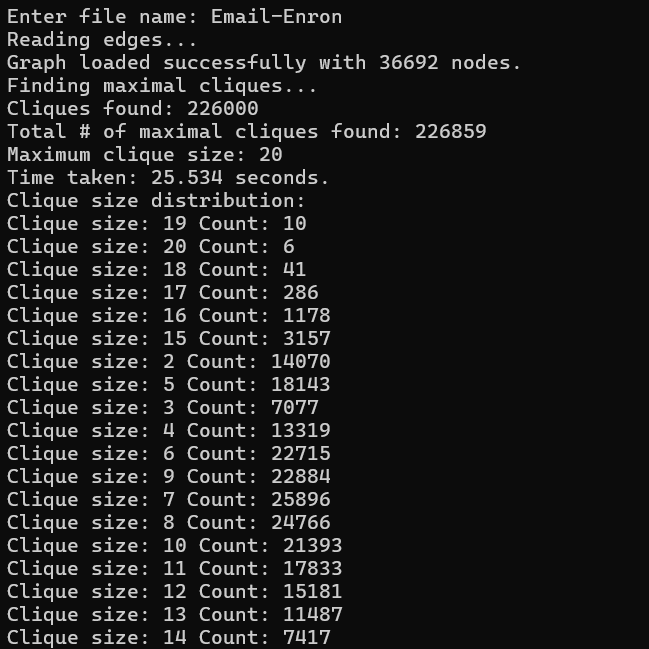


The figure corresponding to the graph dataset given in the paper (Tomita, Tanaka & Takahashi, 2006).



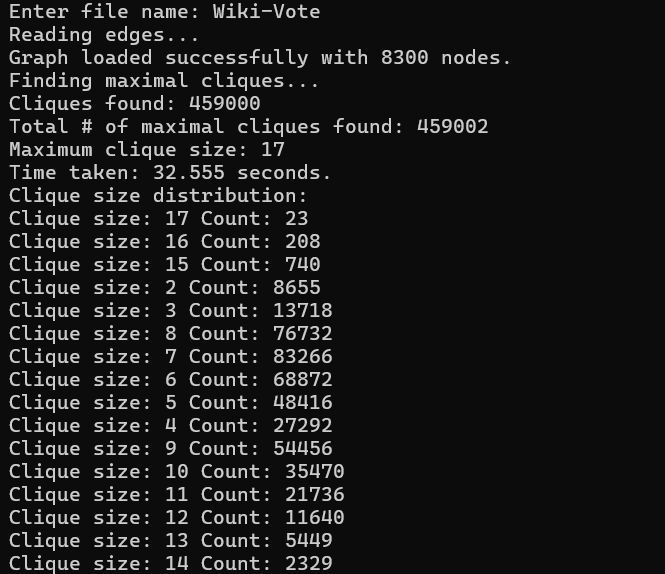
Output of the code for dataset 1 using Tomita’s algorithm.

Dataset 2



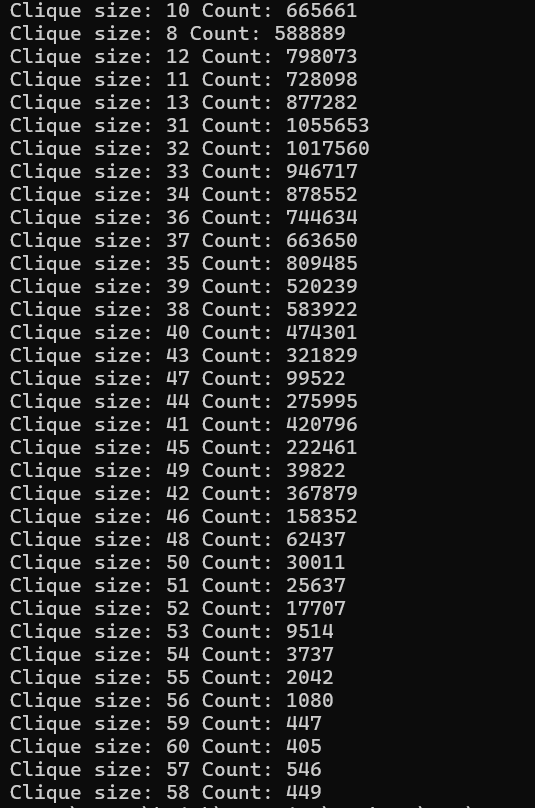
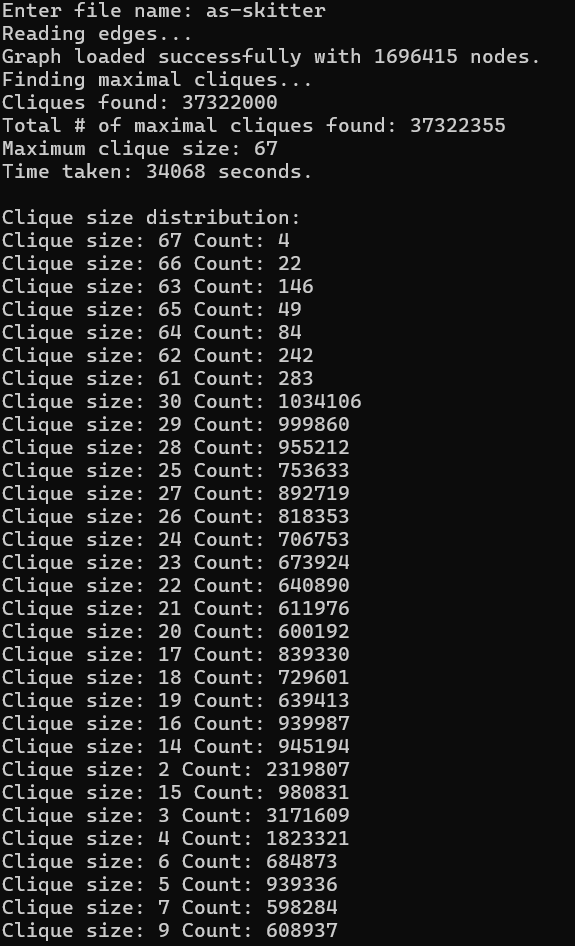
Output of the code for dataset 2 using Tomita’s algorithm.

Dataset 3:



Output of the code for dataset 3 using Tomita’s algorithm.

Dataset 4:



Output of the code for dataset 4 using Tomita’s algorithm.

**Listing All Maximal Cliques in Sparse Graphs in near-optimal Time**

## **Algorithm Overview**

The MaxClique algorithm is designed to find the maximum clique in an undirected graph. A clique is a subset of vertices in a graph where every two distinct vertices are adjacent. The maximum clique is the largest such subset.

## **Key Components**

1. Input: An undirected graph G = (V, E)
2. Output: The size of the maximum clique and the vertices in it
3. Main procedure: MaxClique
4. Recursive subroutine: bonkerboschPivot

## **Algorithm Description**

The MaxClique algorithm employs several pruning techniques to efficiently find the maximum clique:

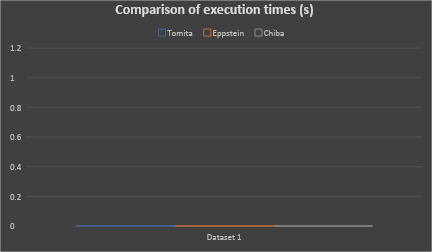
1. Vertex Ordering: Vertices are processed in descending order of degree.
2. Pruning Techniques:
   * Pruning 1: Skip vertices with degree less than the current max clique size.
   * Pruning 2: Only consider neighbors with higher indices.
   * Pruning 3: Skip neighbors with degree less than max clique size.
   * Pruning 4: Terminate if remaining vertices can't form a larger clique.
   * Pruning 5: Only consider high-degree neighbors in recursive calls.
3. Recursive Exploration: The algorithm recursively explores potential cliques, updating the maximum clique size as it progresses.

**CHIBA’s Algorithm**

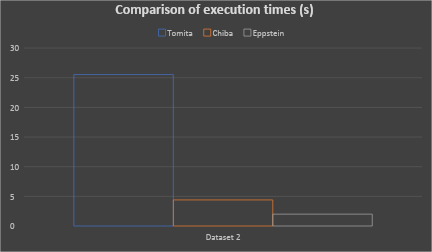
The algorithm follows a recursive approach, exploring vertex neighborhoods to identify and extend maximal cliques. It maintains efficiency through vertex ordering by degree.

1. **Graph Preprocessing:**
   * The vertices of the given graph G are ordered in the ascending order of their degrees
   * Data structures S and T are initialized. They are used throughout the algorithm.
2. **Recursive Clique Extension (UPDATE Function):**
   * The algorithm starts from an initial clique set C and attempts to expand it by adding vertices while preserving the clique property.
3. **Maximality and Lexicographic Testing:**
   * At every recursive call, before adding a vertex to the clique, checks are performed to determine whether the current set forms a maximal clique.
   * The algorithm also ensures that the clique is lexicographically the largest by verifying ordering constraints.
4. **Backtracking and Cleanup:**
   * If a valid maximal clique is found, it is stored/printed.
   * The function backtracks by restoring previous states and continuing the search.
5. **Termination:**
   * The recursive function ends when all possible clique extensions have been explored.

Comparison of execution times for each algorithm



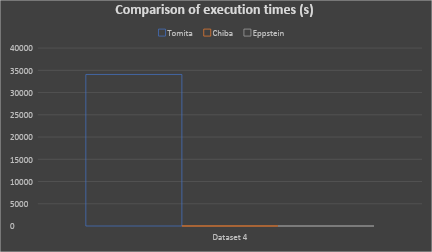
**FIG 6.1**



**FIG 6.2**



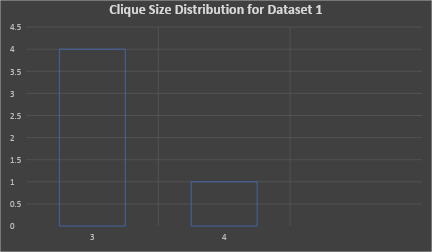
**FIG 6.3**



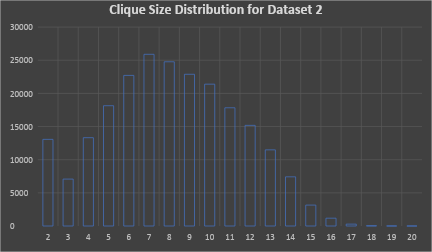
**FIG 6.4**

**FIG 6.1-6.4**: Comparison of execution times in seconds of each of the algorithms for each dataset.

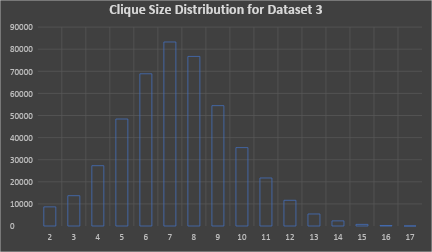
**Clique Size Distribution of all Datasets**



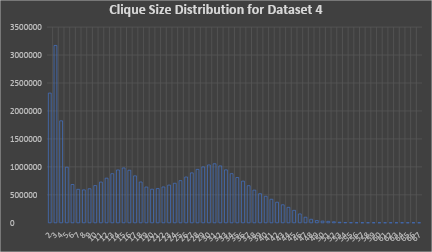
Distribution of Clique Sizes for Dataset 1.



Distribution of Clique Sizes for Dataset 2.



Distribution of Clique Sizes for Dataset 3.



Distribution of Clique Sizes for Dataset 4.