T.C.

ANTALYA BILIM UNIVERSITY

INSTITUTE OF POSTGRADUATE EDUCATION

ELECTRICAL AND COMPUTER ENGINEERING

THESIS PROGRAM

SOCIAL NETWORK ANALYSIS

DISSERTATION

Prepared By

Humair Khan BUGHIO

ANTALYA – 2021

T.C.

ANTALYA BILIM UNIVERSITY

INSTITUTE OF POSTGRADUATE EDUCATION

ELECTRICAL AND COMPUTER ENGINEERING

THESIS PROGRAM

SOCIAL NETWORK ANALYSIS

DISSERTATION

Prepared By

Humair Khan BUGHIO

Dissertation Advisor

Prof. Dr. Cafer ÇALIŞKAN

ANTALYA – 2021

APPROVAL/NOTIFICATION FORM
ANTALYA BİLİM UNIVERSİTY
INSTITUTE OF POST-GRADUATE EDUCATION

HUMAIR KHAN BUGHIO, a master student of Antalya Bilim University, Institute of Post Graduate Education, Electrical and Computer Engineering owning student ID 181212022, successfully defended the thesis/dissertation entitled "Social Network Analysis", which he prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Academic Title, Name-Surname, Signature

Jury Member (Chairman) : Prof. Dr. Cafer ÇALIŞKAN

Jury Member : Ass. Prof. Deniz Gençağa

Jury Member : Ass. Prof. Murat Ak

Date of Submission: ……..

Date of Defense: 28/01/2021

Director of the Institute : ……

DEDICATION AND ACKNOWLEDGMENT

I dedicate this thesis to my parents, friends, and teachers who always supported me to this day of my life.

I am very grateful to my advisor Prof. Dr. Cafer ÇALIŞKAN, who guided and encouraged me throughout the thesis, who opened the doors for me in this research field.

I would also like to thank the jury members **Ass. Prof. Dr. Murat Ak** and **Ass. Prof. Dr. Deniz Gençağa.**

ACADEMIC DECLARATION

I hereby declare that this master's thesis titled "Social Network Analysis" has been written by myself under the academic rules and ethical conduct of the Antalya Bilim University.

I also declare that the work attached to this declaration complies with the university requirements and is my work.

I also declare that all materials used in this thesis consist of the mentioned resources in the reference list. I verify all these with my honor.

04 /01/ 2021

Humair Khan Bughio

ÖZET

SOSYAL AĞ ANALİZİ

Verilen bir çizgede olası maksimum klikleri bulmak, çizge kuramının temel problemlerinden biridir. Bir çizgede maksimal kliklerin numaralandırılması NP-zor bir problem olsa da, bu problem için çeşitli algoritmalar ve yeni yöntemler önerilmiştir. Bazı uygulamalarda mümkün olan en büyük maksimal klikler, yani maksimum klikler, önemli bir rol oynar. Bu çalışma, bir çizgede maksimum klikler bulan yeni bir algoritma önermektedir. Önerilen algoritma, onları bulmak için farklı gerçek dünya veri kümelerine uygulanır. Ayrıca bu çalışma, algoritmanın iki versiyonunu önermektedir; biri tüm olası maksimum klikleri bulur ve diğeri belirli bir grafikte yalnızca bir maksimum klik bulur. Performansı değerlendirmek için, önerilen çalışmanın sonuçları iyi bilinen Bron Kerbosch algoritması ile karşılaştırılır. Sonuç olarak, önerilen algoritmanın Bron Kerbosch yönteminden daha iyi performans gösterdiği görülmüştür.

Anahtar Kelimeler- Sosyal ağlar; klikler; maksimal klik; maksimum klik.

ABSTRACT

SOCIAL NETWORK ANALYSIS

Finding possible maximal cliques in a given graph is one of the basic problems of graph theory. Although the enumeration of maximal cliques in a graph is an NP-hard problem, various algorithms and novel methods for this problem have been proposed. In some application's largest possible maximal cliques i.e. maximum cliques play an important role. This study proposes a new algorithm that finds maximum cliques in a graph. The proposed algorithm is applied to different real-world datasets to find them. Moreover, this study proposes two versions of the algorithm; one finds all possible maximum cliques, and the other finds only one maximum clique in a given graph. To evaluate the performance, the results of the proposed study are compared with the well-known Born Kerbosch algorithm. As a result, it is found that the proposed algorithm performs better than the Born Kerbosch method.

Keywords- Social networks; cliques; maximal cliques; maximum cliques.

Table of Contents

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

BFS	Breadth-first Search
DFS	Depth-First Search
S.N.	Social Networks
BK	Bron–Kerbosch
C.P.	Carraghan & Paradalod
DOCNA	Detection of Overlapping Communities in Networks Algorithm
NP	Non-deterministic polynomial-time
MCR	Maximal Clique Algorithm
LDC	Largest Degree

CHAPTER 1

1 . INTRODUCTION

Over the recent years, there has been a substantial amount of research on models to capture social networks' unique structures [1]. Some of the complex structures are defined on social networks, web networks and biological networks [2]. Despite its simplicity, social network concepts have become increasingly important for some of the applications. Researchers who work on this topic are interested in finding out about the relations among the users in the network. This work is mostly beneficial for obtaining a better understanding of the users, their roles and the structure of the network. It also helps the analyst to foreseen the changes/updates in near future, where possible.

Analysis on social networks is applied on a wide variety of disciplines, from engineering to social science applications. For instance, in social science related studies such works help researchers better understand the individual or group behaviors. Communications or interactions among the members are captured and this is of a great help on understanding the structure of that particular network. Another example may be on e-commerce applications. Analysis of social networks in this particular area gives rise to obtaining a more comprehensive understanding on customers' behaviors. The business owners can shape their services etc. accordingly and provide with suitable resolutions even in advance for their customers.

One may be interested in finding out about the properties of the social networks together with its substructures. These properties and substructures stem mostly stem from the interactions among the users. Consider a group of users in which everyone has some kind of connection to every other one. This particular example determines a unique substructure within the network. If a social network is represented by an undirected simple graph where vertices correlate to the users and edges to their relationships in the network, then that particular group of users mentioned above gives rise to a subgraph which is isomorphic to a complete graph, which is also called as a *clique* in the original graph. More formally, let $G = (V, E)$ be an undirected graph and $C \subseteq V$, then C is said to be a *clique* if any vertex in C is adjacent to any other vertex in C. Moreover, a clique is said to be *maximal* if it is not expandable with another neighboring vertex [3]. Among the maximal cliques, ones with the largest order (or size) are called maximum cliques.

With this definition, one can easily say that finding the maximal or maximum cliques in social network graphs is important when a social network analysis is being conducted [3]. In this scope, some specific applications, in which finding cliques is crucial, are financial network analysis, dynamic clustering of the networks, hierarchy detection in email networks, various pattern matchings in biological systems, computational tasks, and cognitive behavior studies. Some of these real-time applications are being used in large-scale Social Networks (S.N.) to conduct Social Network Analysis (SNA) [4]. Since some of these studies examine the broad and complex social networks, they have long processing time (high time-complexity), so they require some large computational resources to process large-scale datasets [5]. In this sense, designing more time-efficient algorithms for finding cliques is important.

In this thesis, we introduce an algorithm that finds the maximum cliques with lower time complexity in large-scale S.N.'s. Some well-known methods find maximal (maximum) cliques by traversing the whole graph from node to node. In our approach, we focus on the potential ones instead of traversing all vertices. This approach results in providing a resolution with a lower time complexity.

1.1 Motivation

In the current era, the problem of finding cliques in a large-scale dataset is one of the main problems in graph theory. From a theoretical or practical perspective, it becomes more important in various real-life problems to find cliques provided that the network is modeled via graphs. In parallel, some research areas such as data mining has gained much more importance in recent years. With this effort, some algorithms have been developed to find the cliques, especially the maximal cliques. One replaced another over time as some algorithms with lower time complexity have been developed. The proposed study improves the performance for finding the maximal cliques compared to some of the existing celebrated methods even if there exist some faster algorithms. In that sense, our goal in this study has been to propose a more efficient algorithm than the current well-known examples. The outcome seems to be successful partially.

1.2 Thesis Organization

In this thesis, Chapter 2 describe preliminaries on Graph theory and Graph Algorithms. Moreover, Methods, traditional approaches and briefly describe the clique’s

problem in the chapter 3. Furthermore, a brief description of the dataset and the proposed algorithm are explored in Chapter 4. Then, Chapter 5 explains the results of the proposed approach, and finally Chapter 6 discusses the results obtained from the proposed method applied on some datasets and includes future work.

CHAPTER 2

2 PRELIMINARIES

This chapter introduces some basic terminology including definitions, terms etc. used in graph theory. In general, graphs are used theoretically to model some applications so one can describe the existing original problem on this graph with graph theoretical tools. For instance, in computer science, some data structures are generated in the form of a particular graph called tree in which some methods related to data mining, image segmentation, clustering, image capturing and networking find some base and become applicable. In a similar way, graph structures are used to model network topologies. With this, some methods such as graph coloring etc. can be applied for some problems to create resolutions. Some well-known problems such as traveling salesman problem become easy to describe and even easy to be handled under some assumptions [6].

In what follows we discuss about some basic notions in graph theory.

2.1 GRAPHS

A graph G is a collection of objects called vertices $V = V(G) = \{v_1, v_2, \dots, v_n\}$ together with a set of edges $E = E(G) = \{e_1, e_2, \dots, e_m\}$. Vertices are sometimes called as nodes or points. If $e_i = v_k v_l$ is the edge between the vertices v_k and v_l, then these two vertices are said to be adjacent or neighbors. A graph with no self-loops or parallel edges is called a simple graph. In a graph G, the number of vertices is same as the order of G and number of edges the size of G.

For a given graph G, a subgraph H is graph where $V(H)$ is a subset of $V(G)$ and $E(H)$ is a subset of $E(G)$. In a graph G, a u-v walk is a sequence of vertices in which the consecutive vertices are adjacent. A graph G is said to be connected if there always exists a u-v walk for any two random vertices u and v in G.

Two graphs are said to be isomorphic if they have the same structures. This implies the same order and size as well as having the same graph structure. One way to show that a graph G is isomorphic to another graph H is to show that one could be obtained from another by just relabeling the vertices or/and repositioning the vertices without breaking the original edges. If two graphs are not isomorphic, then they are said to be non-isomorphic. See Figure 1 for two isomorphic graphs.

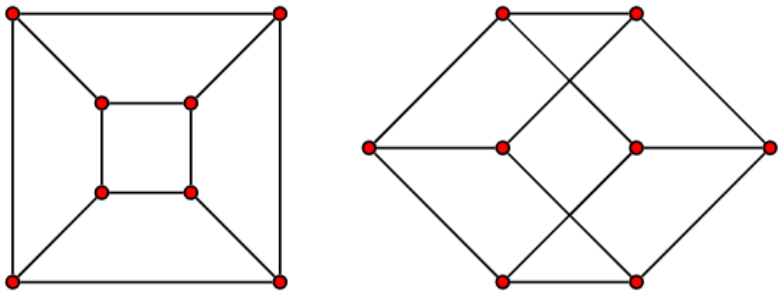

Figure 1 Two isomorphic graphs

A graph G is said to be undirected if the edges are missing directions. However, if the edges are the ordered pairs of vertices i.e. some directions are applied on the edges, then the graph is called a directed graph or digraph. See Figure 12 for examples of directed and undirected graphs. The following graph means an undirected graph unless otherwise indicated.

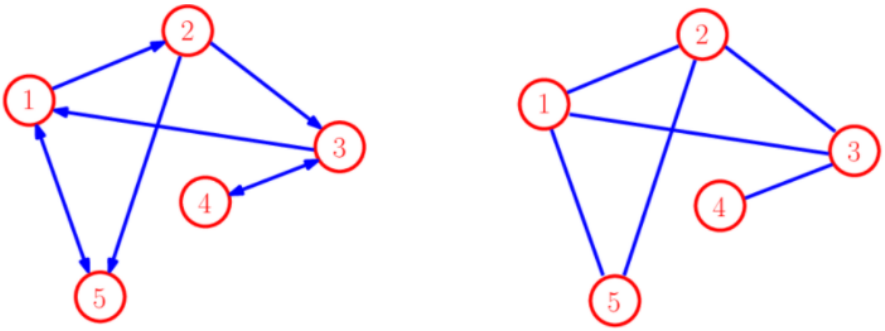

Figure 2 Directed and Undirected Graphs

As defined above, given a graph $G = (V, E)$, the two different nodes u, $v \in V$ are said to be neighbors, if $(u, v) \in E$. When the graph G is directed, the terms of outgoing or incoming neighborhoods come into action depending on the direction defined on the edge.

In an undirected graph $G = (V, E)$, the degree of a vertex u is the number of neighboring vertices in the graph i.e.

$$deg(u) = |\{ v \in V : uv \in E \}|.$$

In the case of a directed graph, the terms of in-degree and out-degree are determined by the number of incoming edges and number of outgoing edges, respectively. The degree sequence of a graph is a list of non-negative integers that are consist of all the degrees in the graph.

A complete graph is one of the basic graphs in which any two vertices are adjacent. A complete digraph is a graph that connects each pair of separate vertices with a pair of specific edges. For a given graph G, if one of its subgraphs, namely H, is isomorphic to a complete graph, then H is said to be a clique in G. A clique is said to be maximal if it is not expanded with one more neighboring vertex. Among the maximal cliques in G, one with a greatest order is called a maximum clique. For a given graph, the problem of finding the maximum cliques is called the Maximum Clique Problem.

2.2 GRAPH ALGORITHMS

In graph theory, another important notion is the topic of graph algorithms. When one comes up with a representation of a problem with a graph G, then a solution requires mostly some graph theoretical algorithms defined on G. In the following, we give basics of the graph algorithms with focusing on few approaches.

An algorithm is a finite sequence of clearly defined computer instructions for solving a problem or performing the computation. Algorithms are always used for evaluation, data processing, computational analysis and other calculation tasks. Table 4 below shows some of the well-known algorithms that are used to resolve graph-related problems [6].

Table 1 Graph Algorithms

1	Breath-First Search Algorithm (BFS)
2	Depth-First Search Algorithm (DFS)
3	Dijkstra's Shortest Path Algorithm

4	Bellman-Ford Algorithm
5	Floyd Cycle Detection Algorithm
6	Prim's Algorithm
7	Kruskal's Algorithm
8	Kahn's Algorithm
9	Ford-Fulkerson Algorithm
10	Edmonds-Karp Algorithm
11	Hopcroft-Karp Algorithm

In Breadth-first search (BFS), one can start at particular vertex and goes to its neighbors at the current depth before visiting other vertices in the next level. If it is applied on a tree structure, the algorithm visits the vertices depending according to the distance from the root of the tree. It is an algorithm to explore or search the tree or a graph in general.

BFS computing is synchronous by levels, and an algorithm traversal is divided into varying computational steps simultaneously. Each level has a frontier, which is the "Curr" in algorithms. The frontier is the level where all active vertices are found. The current frontier produces the next frontier during computational at various levels, which is the algorithm's *next phase*. BFS starts with the root vertex is called v_r. When the frontier is empty, it terminates when each level completes. A synchronization process is required [7].

Table 2 Basic BFS Algorithm

Input: Graph $G = (V, E)$, a vertex $v_r \in V$

for all $u \in V$ do
    $\text{distance}(u, v_r) = \infty$
$distance(v_r, v_r)$ =0
$queue = [v_r]$
while $queue$ is not empty **do**
    $u = eject(queue)$
    for all edges $uv \in E$ **do**
        if $distance(v, v_r)$=∞ **then**
            $inject(queue, v)$
            $distance(v, v_r) = distance(u, v_r)$+1

Figure 3 shows below how the BFS traverses over the graph.

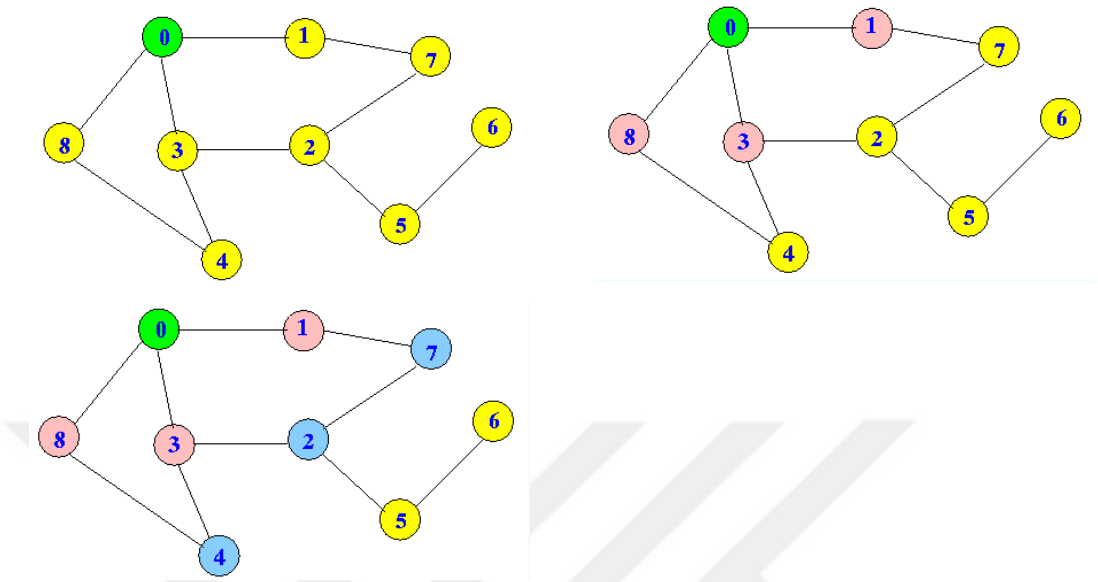

Figure 3 BFS Traversing the Graph

In Depth-first search (DFS), one can start at a particular vertex and continues visiting the vertices as far as possible along each branch before retracing back. It is used for searching and using a stack data structure. DFS's ability to find each node that has not been visited makes finding the optimal solution in some problems faster such as the longest path problems. Searching or tracking may solve general problems. Some applications of this approach is to find paths between vertices, to detect cycles and to sort topologically.

Table 3 Basic DFS Algorithm

Input: Graph $G = (V, E)$, a vertex $v_r \in V$

$visited(v_r) =$ **true**
$previsit(v_r)$
for each edge $v_r u \in E$ **do**
    if not visited(u) **then**
        $explore(u)$
$postvisit(v_r)$

In Table 3 above, the pseudo-code for the DFS algorithms illustrates how DFS is working and how it traverses the graph. It visits each vertex from depth. Figure 4 shows

below how the DFS traverses over the graph for few steps. Moreover, Figure 5 explains how both BFS and DFS algorithms work on a tree.

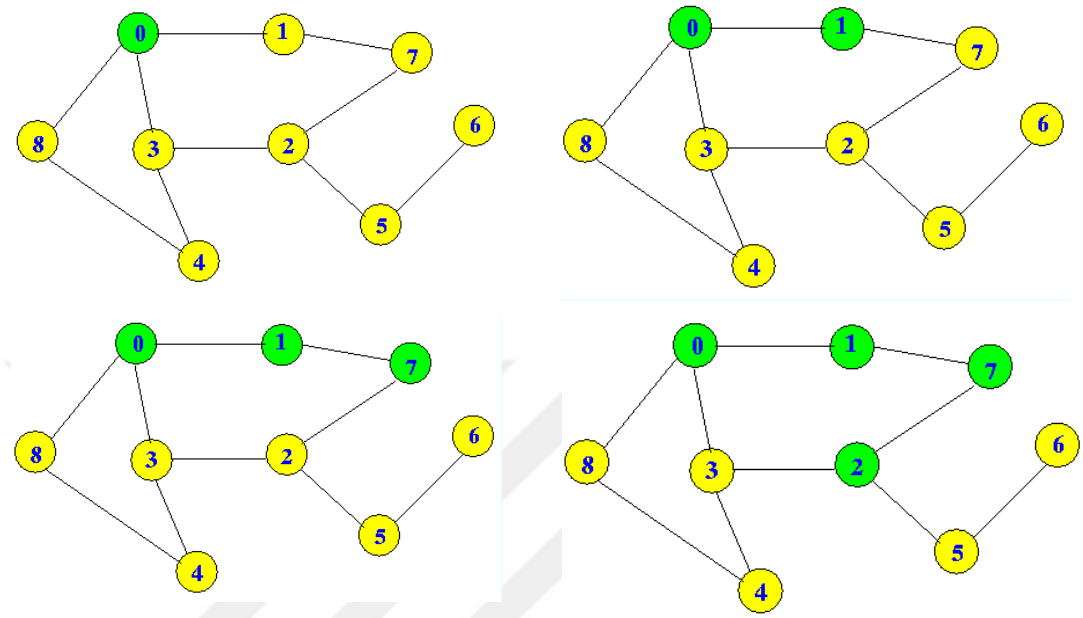

Figure 4 DFS Traversing the Graph

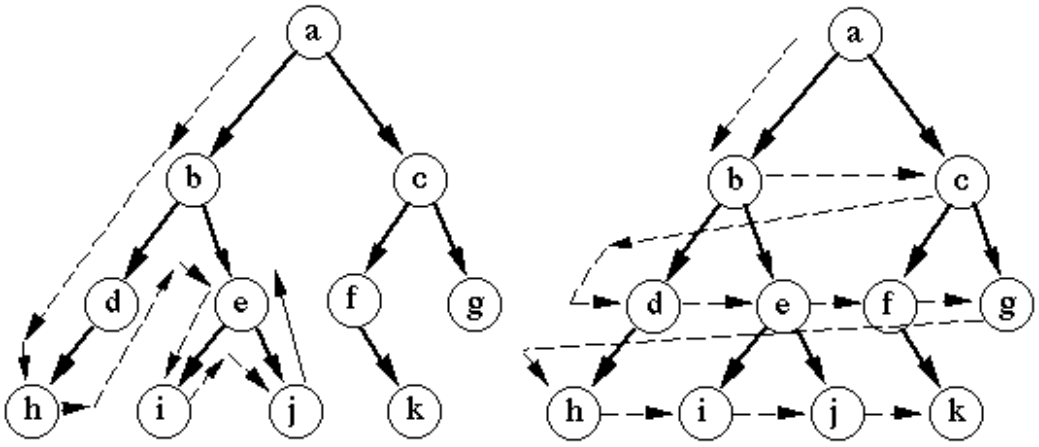

Figure 5 DFS vs BFS algorithms applied on a tree structure

In this study, the computational sections are handled in Python language. In particular, a Python language package named NetworkX is used for network and network algorithm discovery and analysis. The core package includes data structures for various network types or graphs, including basic graphs, directed graphs, and parallel-edged and self-loop graphs. Due to its simplicity, this package is suitable for representing networks in various fields [8].

CHAPTER 3

3 METHODS ON THE CLIQUE PROBLEM

In the current decade, as a part of social network analysis, finding cliques are one of the common problems in graph theory. In this area, social network analysis (SNA) has a group of commonly used methods and different techniques to analyze the large scaled data-sets. Recently, there have been many algorithms, mostly user-based algorithms, developed. With this motivation, this section discusses about some well-known clique detection methods. Some experimental results are described in this chapter as well.

3.1 MAXIMUM CLIQUE PROBLEM

In a graph G, a maximum clique is a clique with the greatest order. In general the problem of finding the maximum cliques an NP-hard problem [9]. But even an approximate solution is complicated to find. In this problem, branch and bound is one of the most common search efficiency techniques. In 2007, Tomita built a related algorithm to find the maximum cliques based on coloring and vertical ordering [10]. This algorithm is known as MCR and has proven faster than other algorithms to find the maximum cliques. According to another report published in 2009, MCR is decent but not quick enough for large and complex graphs [11]. A new MSC algorithm was introduced to solve this regression, incorporating new ordering and coloring techniques into MCR. The experimental results show that this new method is faster than the original one for high-density graphs and sparse graphs. In that proposed method, it was aimed to implement efficient graph coloring techniques to make a list of the colored graph with minimum coloring to find the clique from the large set of a graph. Its approach is better than heuristic approaches on runtime [12]**.** However, it is still an open problem of finding a more efficient algorithm for finding the maximum cliques.

Figure 6 shows below a connected graph of order 23. Within this graph, the induced subgraphs with vertices $\{a, b, c, d, e\}$, $\{l, m, n, o\}$ and $\{w, v, x\}$ are isomorphic to complete graphs of order 5, 4 and 3, respectively. However, one with vertices $\{a, b, c, d, e\}$ is of the greatest order and a maximum clique.

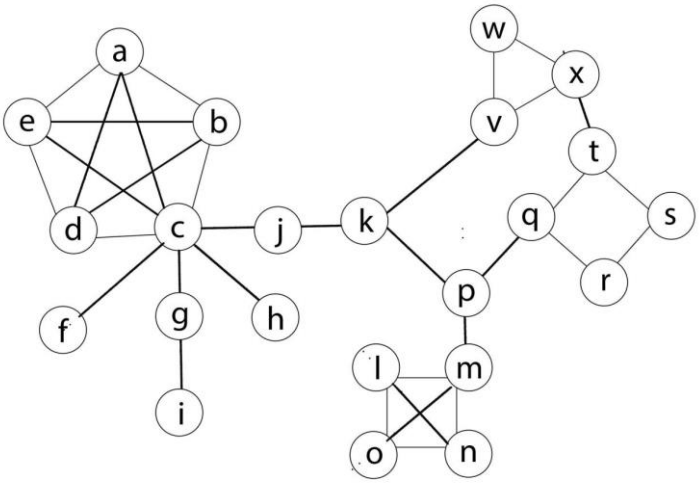

Figure 6 A Graph of order 23

3.2 BRON-KERBOSCH (BK) ALGORITHM

Many techniques are used to resolve the clique detection problem in large-scale social networks, so we discuss one of the most well-known methods, namely Bron-Kerbosch. For finding the maximal cliques in an undirected graph, the Bron–Kerbosch Algorithm is among the most powerful and efficient algorithms. In what follows, we describe how it runs in detail.

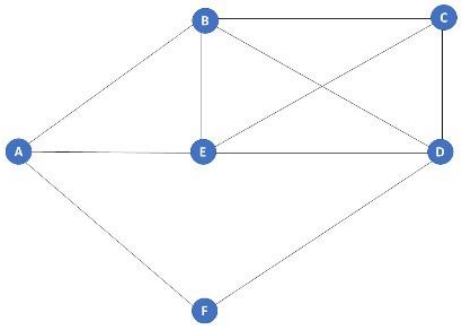

Figure 7 A Graph G of order 6

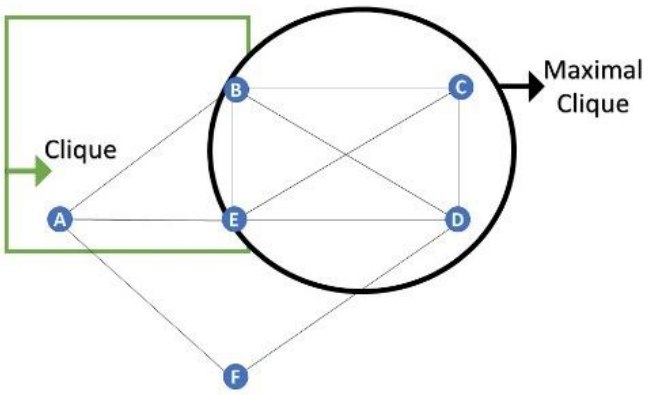

Figure 8 Maximal Clique in G

In 1973, Coenraad Bron and Joep Kerbosch designed the algorithm and they basically built a backtracking algorithm using a search tree that offers a solution for the clique problem from a root to a leaf. In that approach, they used a branch-and-bound algorithm to decrease the number of unexplored branches that cannot contribute to a clique. To understand the process, we need to understand how the algorithm works. The algorithm works with three sets of nodes (R, P, X). Set R is known as the upcoming clique and P is the node that connects with all other nodes in the set R that is to be used for next. X is the set that has already been used to find the maximal clique or already processed nodes of the list. It important that all nodes linked to each R node are either P or X. The purpose of X is to avoid repeatedly disclosing the same maximum clique by updating $P = P - \{ui\}$. The algorithm tests if set X is empty; if X is non-empty, for preventing disclosing clicks that are not maximal, the X nodes can then be added to R.

The BK algorithm uses the recursive backtracking techniques to search all maximal cliques from graph G. Moreover, the algorithm works on sets (R, P, X). It shows all maximal cliques are directly connected with the set R; also, some in the P but not of them in X. P and X are disjointed pairs, which are joined by the vertices forming the cliques when added to R. The pseudocode for this algorithm is given below.

Table 4 BK Algorithm

```
BronKerbosch(R, P, X):
    if P and X are both empty:
      report R as a maximal clique
    for each vertex v in P:
      BronKerbosch(R ∪ {v}, P ∩ N(v), X ∩ N(v))
      P := P \ {v}
      X := X ∪ {v}
```

For graphs with several non-maximal cliques, the simple form of the algorithm mentioned above is unsuccessful. For any clique, it makes a recursive call, whether maximal or not, to save time and will enable them to backtrack algorithm in search Faster branches that do not contain maximum cliques. This version of the algorithm is given below.

Table 5 BK Algorithm

```
BronKerbosch(R, P, X):
  if P and X are both empty:
    report R as a maximal clique
  choose a pivot vertex u in P ∪ X
  for each vertex v in P \ N(u):
    BronKerbosch(R ∪ {v}, P ∩ N(v), X ∩ N(v))
    P: = P \ {v}
    X: = X ∪ {v}
```

The BK algorithm has the following time and space complexities:

i) Time Complexity: $O(3^{n/3})$ worst-case time to complete.

ii) Space Complexity: $O(n^2)$ worst-case space to complete.

3.3 The CLIQUES Algorithm

Etsuji Tomita et al. introduced the CLIQUES Algorithm which is based on two significant algorithms, the first one is the graph traversing algorithm known as Depth-First Search (DFS) and the second one is the Bron-Kerbosch algorithm, which is faster in finding the cliques or maximal cliques in the large dataset from social media networks. In which pruning methods are employed as Bron-kerbosch algorithm. All are produced in a tree-like shape producing maximum cliques. Moreover, G = (V, E) is a given graph the DFS algorithm is used to find the maximal cliques, and the global variable Q is used to store the set of vertices that have all subgraph. In the start, the variable Q is said to be empty, and expends Q as a recursive procedure. The recursive method is to apply Q to EXPAND to V and its following subgraphs to find the subgraphs until finding the maximal subgraph.

Let Q = {p1, p2,….,pd} is the set vertices of complete subgraphs.

$$SUBG = V \cap \Gamma(p1) \cap \Gamma(p2) \cap \cdots \cap \Gamma(pd),$$

Where SUBG begins as V and Q begin ∅ at the initial level. Apply to EXPAND to SUBG to find the larger complete subgraphs. If SUBG = ∅ then Q is a maximal clique, or Q is a maximal complete subgraph. Consider the smaller subgraphs G (SUBGq) consisting of the vertices that added a new set of vertices.

$$SUBGq = SUBG \cap \Gamma(q)$$

for all q ∈ SUBG, extend recursively to $SUBG_q$ to reveal larger complete subgraphs containing Q ∪ {q}. This model can be characterized by the following forest search, which is the set of search trees that are exactly the same as the vertices of V of graph G. (V, E). therefore, the q ∈ SUBG and every vertex in the $SUBG_q$ are the child vertex of the q. Thus, a clique or subgraph is a set of vertices along a path from the root to any search forest vertex. Now define two ways to prune the search tree, which are used by the Bron-Kerbosch algorithm. As previously mentioned, set SUBG (≠ ∅) a set of vertices and an ordered. Next, proceed to find maximal cliques at each stage from the vertices in that set.

First, we need to measure a set of vertices (called “FINI”) that the algorithm has already processed. So, we then denote by

$$CAND: CAND = SUBG - FINI$$

Hence, after that, we have

$$SUBG = FINI \cup CAND\ (FINI \cap CAND = \emptyset).$$

FINI = ∅ at the starting. Consider the graph G (SUBGq), where SUBGq is defined here. Also, let.

$$SUBGq = FINIq \cup CANDq\ (FINIq \cap CANDq = \emptyset),$$

Were

$$FINIq = FINI \cap \Gamma(q)\ and\ CANDq = CAND \cap \Gamma(q).$$

Next, there are only vertices in CANDq that belong to cliques that can be extended to find new larger cliques. Moreover, consider a vertex u in SUBG. Suppose all maximal cliques containing $Q \cup \{u\}$ are formed. Thus, any new maximal clique containing the Q. just not in Q∪ {u}, get at least one vertex. $q \in SUBG - \Gamma(u)$. That is because if Q is found in a complete subgraph $R = (Q \cup S) \cap (SUBG - \{u\})$

With

$$S \subseteq SUBG \cap \Gamma(u),$$

then $R \cup \{u\}$ is a complete subgraph that includes a larger subgraph of the graphs, but R is not a maximal. Any new maximal clique can be identified by extending $Q\ to\ Q \cup \{q\}$ such that $q \in SUBG\ \Gamma(u)$ and generating all the cliques containing $Q \cup \{q\}$.

The only search subtrees also take into account the previously mentioned pruning process, and they are from the vertices to be extended in $(SUBG - SUBG \cap \Gamma(u)) - FINI = CAND\ \Gamma(u)$. Here, to reduce the $|\ CAND\ \Gamma(u)\ |$, Selecting a vertex $u \in SUBG$ as the one which maximizes $|\ CAND \cap \Gamma(u)|$ [13].

3.4 C.P. ALGORITHM (Carraghan & Paradalos 1993)

In 1993 an algorithm named C.P. was introduced by Carraghan and Paradalos. This algorithm provides with a solution for the clique problem. To do this, C.P. uses two basic vertices, C is for (Clique), and P for the set of candidate vertices. The clique (C, P) function is defined by two essential components: branching and bounding. The algorithm is obtained by selecting the original vertex from several vertices in the existing clique [14].

Later in 1990, Babel and Tinhofer have proposed another algorithm. This algorithm is based on purining and bounding to find the maximal cliques. In the bounding strategy, it's working same as the C.P. algorithm that we already discussed above. The algorithm uses the same number of vertices on the set of P as C.P. Whereas, in the pruning, the algorithm aims to provide an efficient upper bound with the use of a greedy DASTUR technique [14]

A study in 2012 presented an overlapping group discovery algorithm. Every node vote in the community was combined with the last local community in a global set using a level propagation algorithm [15]. In 2013, Tooth et al. studied the modularity of the Group system by means of a clique percolation process. In 2008, Clara Pizzuti launched a genetic algorithm that uses a node-based clustering principle [16]. In 2009, Clara Pizzuti introduced genetic algorithms which exploited edge clustering [17]. Edge clustering is discovered by using the line graph. In 2011, Cai et al. proposed genetic algorithms to identify populations that overlap [18]. In 2013, Dickinson et al. suggested an overlapping population identification algorithm using genetic algorithms [19]. This algorithm recognizes populations and then connects communities with overlapping communities.

3.5 FEMC ALGORITHM

In this approach, there is a proposition on a novel depth-first search algorithm to find all maximal cliques in an undirected graph. Numerous real networks, random graphs and demonstrate the efficiency of our experimental experiments. Computer tests demonstrate that our algorithm in these graphs is better than the Bron-Kerbosch Algorithm [20].

In a graph G, the FEMS is the algorithm that has a process to search given nodes V_i, for $\forall_{Vj}$ belongs to $N_{L.}$ (v_i) the process of searching, first of all, finds every node V_k belongs to Δ (V_i, V_j) that generate triangle with V_i and V_j. V_i and V_j are the nodes and further recursively $V_m \in \Delta$ (V_i, V_j), $m \neq k$ that makes $V_m \in \Delta$ ($V_{i,}$ V_k) with the combination set that forms triangle Δ (V_i, V_j). $V_m \in \Delta$ ($V_{i,}$ V_k), $V_m \in \Delta$ (V_i, V_j). V_i, $V_{j,}$ $V_{k,}$ and V_m are the nodes that make up maximal cliques with the size 4. Finally, a search tree is obtained from a single starting node is called root node V_i. Furthermore, internal nodes on the tree are a clique, and others are leaf nodes that are all possible maximal cliques, including the V_i node. In the particular graph G, Lema-1 has defined whether the nominee is a maximal clique or not

Lemma 1. form the given graph G, any nominee clique Gs, Gs will be maximal clique if and only if $\forall \in V(G)$ and nodes $u \in N(v)$-$V(Gs)$ that makes $V(Gs) \subset N(u)$ but not exist.

Pruned Rule 1. A nominee clique G_s, the smallest node is often selected to decide whether or not G_s is a maximal clique.

Pruned Rule 2. If nodes in the current clique have the same label as nodes of the expanded clique in a specific search tree, expansion suspends and responses recursively.

Graph G, enumerating maximum cliques in the V_i search tree is independent of the search tree process Vj. The following lemma is then proposed here.

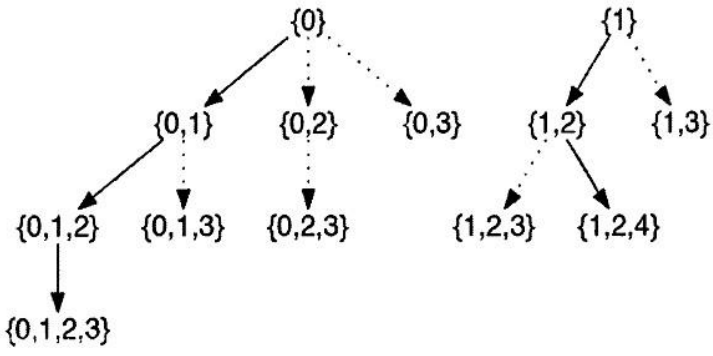

Figure 9 FMEC Maximal Cliques

Lemma 2. The enumerating of all possible maximal clique of an N node graph is a task that can be divided into N subtasks that is mutually independent. The subtask lists the cliques aligned with each node.

For example, in Figure 5 dotted arrows. 2 implies certain lines should be cut so that you can quickly locate maximum cliques, For example, set of {0,1,2,3} and set of {1,2,4}, on the solid arrow, in compliance with the pruned laws.

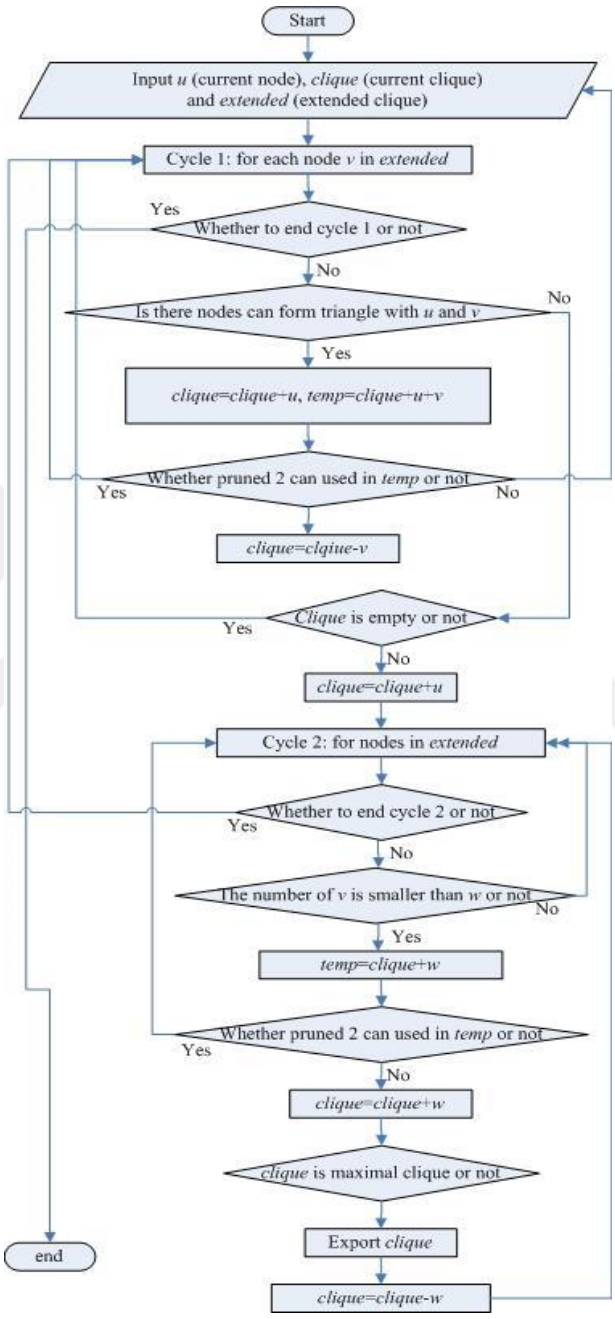

Figure 10 Displays the FEMC flowchart

Moreover, we applied ourselves to publicly accessible data sets, which led to four real-world networks being selected: Karate, Dolphins M, and Football. For maximal clique enumeration, FEMC is as fast as B.K. The benefit of FEMC over B.K. is instantly seen

on the large network. B.K. is roughly twenty times slower than FEMC to count entirely the cliques in a network. In these results, we prove that our algorithm is efficient in finding all maximal cliques.

Table 6 Performance Comparison in Datasets

Networks (n, m)	Karate (34,78)	Dolphins (62,159)	Football (115,613)	Power (494,6594)
C	25	53	185	464
FEMC	93(µs)	265(µs)	6.22(s)	19.73(s)
BK	109(µs)	296(µs)	2.07(s)	450(s)

3.6 DOCNA

In this method, a novel algorithm called DOCNA identifies overlapping network groups of communities based on maximum cliques. DOCNA accepts a version of updated B.K. algorithm. In reality, it was added a task process that allows each "Lost Node" part of a detected community. The proposed algorithm also uses a particular method of creating a Click-Graph to finds communities. Analysis results on simulated and actual networks with various dimensions and overlapping frequencies demonstrate the efficacy of a proposed approach to identify dynamically overlapping structures in the population. The author suggests a static DOCNA algorithm, which is based on the idea of the maximum cliques, to remove group structures. There are three main phases in the process of the algorithm. The first stage consists of extracting the full cliques using an updated B.K. algorithm variant. DOCNA retains the positive efficiency advantages of B.K. and uses "Lost nodes" to enhance the extraction process of cliques. The second step consists of constructing a Click-Graph, where a click is represented as node, and edge is a link. Finally, the final step involves removing populations from the Breadth-First Search (BFS) algorithm theory [21].

DOCNA algorithm aims to collect all the groups that form a network, as defined in the algorithm1. DOCNA uses two parameters: graph $G = (V, E)$ and which specifies the network portrayed by the graph. The set of nodes is V and E is the edge set, and k is to defend the minimum number of cliques in the extraction process, and an integer value

equivalent to 4. As a result, proposed algorithm returns populations that form the network of the communities. Three main phases characterize it.

The method requires phase one to extract the cliques from given graph G with three different pre-determined phases in the algorithm 2. Next, the approach attempts to define all of the maximal cliques by using the Bron Kerbosch algorithm's third version implementation. But the problem with this algorithm is the presence of what is known as "Lost nodes." In truth, these nodes are vertices which no clique derived from this version of the B.K. algorithm, and hence no group can be allocated. The second stage of this phase, therefore, is to remove its "Lost Nodes." The last step is to follow the range of full clicks within the first step, as initial preexisting cliques. After that, the clique associated with the highest assignment coefficient must be allocated to each "Lost Node." use Formula 1. When an appropriate coefficient has been determined.

$$CA(node, clique) = \frac{|\mathrm{N}(\mathrm{node}) \cap \mathrm{clique}|}{|\mathrm{clique}|}$$

The method requires second phase which have second step involves constructing a Click-Graph using some particular techniques where a vertex represents a clique and an edge represents a relation. The formula 2 overlapping function F_c measures the overlap value used to determine whether or not the edge is located.

$$F_c(clique_i, clique_j) = \frac{V_c}{\max\left(|Clique_i|, \left|Clique_j\right|\right)}$$

The method requires a few different stages to get finished, the last of which is applying the BFS principle on the network communities. This algorithm uses file s and is based on three key steps. Next, the file *s* is routed through a *curr* in the beginning node. Secondly, the algorithm visits and threads all the vertex neighbors when they are not present or already transferred to this file. Thirdly, we delete the head of this file and, as long as possible, begin from the second vertex. The file is as long as it is not completed or empty. This algorithm allows us to discover.

Algorithm 1: DOCNA

Data: *Graph* $G(V, E)$, *Clique size* k
Result: *Communities*

Begin
  $Cliques \leftarrow Extraction\_Cliques(G, k)$
  $C_G \leftarrow Cliques\_to\_Graph(Cliques)$
  $Communities \leftarrow Find\_Communities\_BFS(C_G, Cliques)$
  return *Communities*
End

Figure 11 Algorithm 1 DOCNA

Algorithm 2: Cliques_extraction

Data: *Graph* $G(V, E)$, *Cliquesize* k
Result: *Cliques*

Begin
  $Init_{cliques} = BronKerboschDP(G, k, R, P, X)$
  for ***each*** $clique \in Init_{cliques}$ **do**
    for ***each*** $node \in clique$ **do**
      if $(node \notin NC)$ **then**
        $NC \leftarrow NC \cup \{node\}$
      end
    end
  end
  $NT \leftarrow V \setminus NC$
  $Cliques = Init_{cliques}$
  for ***each*** $node \in NT$ **do**
    if $| N(node) | \neq 0$ **then**
      for ***each*** $clique \in Cliques$ **do**
        $c \leftarrow CA(node, clique)$
        $L \leftarrow L \cup \{c\}$
        $Max_{coef} = max(L)$
        $Val_{coef} = L.index(Max_{coef})$
        $Cliques[Val_{coef}] = node$
      end
    end
  end
  return *Cliques*
End

Figure 12 Algorithm 2 DOCNA

CHAPTER 4

4 DATASET DESCRIPTION AND THE PROPOSED METHOD

In the following, we describe the datasets used in this study and discuss the proposed method.

4.1 Random Graph Datasets

This section discusses the datasets used in the proposed research. The graphs are generated randomly by using the NetworkX package in Python. Each graph consists of different number of nodes and edges. The number of nodes and edges are between (23-100) and (41-1000) respectively. The number of nodes and edges generated by random graphs are shown in the table below.

Table 7 Random Graph with Nodes and Edges

	Nodes	Edges
Random Graph -1	23	41
Random Graph -2	46	83
Random Graph -3	50	600
Random Graph -4	100	800
Random Graph -5	100	1000

4.2 Social Datasets

In this study, three datasets of social networks are used. These datasets are as follow: 1) Politician dataset 2) Media Communities 3) Government. All of these are genreated from Facebook pages under disticnt types. In all, nodes represent the pages and edges the mutual likes between those pages. The number of nodes and edges are shown in the table below.

Table 8 Datasets with number of nodes and edges

Dataset Name	**Nodes**	**Edges**
Politicians	1250	2036
Media Communities	307	237
Government	2553	8406

4.2.1 Politician dataset.csv

This dataset are related to different pages among the politician category pages on Facebook. There is a link between different pages if there exist mutual likes. The dataset contains 1250 nodes, and 2036 edges. The sample data is illustrated in the follwong table.

Table 9 Politician dataset sample entries

Node_1	Node_2
18	24
128	29
18	63

4.2.2 Media Communities.csv

In this dataset, different pages among media communities on Facebook are represented. Links among pages represent mutual likes. It contains 307 nodes, and 237 edges. The sample data is illustrated in the following table.

Table 10 Media Communities dataset sample entries

Node_1	Node_2
16	16
19	32
338	34

4.2.3 Government dataset.csv

Similarly, the dataset shows the structure of different pages among government communities on Facebook and how they are linked to together. Links again represent mutual likes. The dataset contains 2553 nodes, and 8406 edges. The sample data is illustrated in the follwong table.

Table 11 Government dataset sample entries

Node_1	Node_2
51	62
3508	63
3508	66

4.3 The Proposed Algorithm

This section describes the methodology of the proposed study. The main purpose of the study is to find the maximum cliques in a given graph. There exist two versions of the proposed method. In the first, all possible cliques are detected in a given graph and second version detects one maximum clique in a given graph. The proposed algorithm comprises of the following steps.

Proposed Algorithm: First version

Step 1. Find the neighbors of all nodes from the graph.

Step 2. Sort the nodes in descending order with their respective degrees and store them in a list with their neighbors.

Step 3. Select a node with a highest degree along with its neighbors.

Step 4. Check the connection among all the neighbors of each node from step 3.

Step 5. In this step, if all the neighbors from step four are connected then it is a clique and algorithm will move to next node in step three to find more cliques. If the algorithm cannot find the clique at this step it will also move toward next node in step three.

Proposed Algorithm: Second version

Step 1. Find the neighbors of all nodes from the graph.

Step 2. Sort the nodes in descending order with their respective degrees and store them in a list with their neighbors.

Step 3. Select a node with a highest degree along with its neighbors.

Step 4. Check the connection among all the neighbors of each node from step 3.

Step 5. In this step, if all the neighbors from step four are connected then it is a maximum clique in the graph and the algorithm terminates.

4.4 The Methodology of Algorithm

The proposed algorithm aims to extract all possible cliques. It takes the graph $G = (V, E)$ as an input where V is the set of nodes, and E is the set of edges. The following table illustrates the methodology of the algorithm.

Table 12 Proposed Algorithm

Proposed Algorithm V -1

Input:

$G = (V, E)$: a graph

v_r : first node to start

Output:

Clique

Begin:

$node\_list$: contains all the nodes in the list

$neig\_list$: contains all the neighbors of each node

$sorted\_list$: contains all the nodes from highest degree to lowest

1. $\boldsymbol{for}\ Next \leftarrow graph\_node$
2. $\boldsymbol{for}\ i \leftarrow Next$:
3. $\boldsymbol{for} \leftarrow i$
4. $neig\_list = v_r + neighbors\ of\ v_r$
5. $sorted\_list = sort(neig\_list)$
6. **end**
7. **end**
8. **end**

After completing the above phase, the study proposes two different algorithms. The first algorithm finds all possible cliques from the given graph. The pseudocode of this version is given below.

Table 13 Proposed Algorithm Version 1

Proposed Algorithm V-1

Input:

$G = (V, E)$: a graph

v_r : first node to start

Output:

All possible cliques

Begin:

$sorted\_list$: contains all the nodes from highest degree to lowest

$neig\_list$: contains the neighbors of v_r with neighbors

$clique\_list$: contains all possible cliques

1. ***for*** *k in sorted_list:*
2. | *neig_list initialization*
3. | ***for*** *node(v_r) in k:*
4. | | *neig_list = node(v_r) + neighbors of v_r*
5. | |
5. | | *first index = contains node with neighbors*
6. | **end**
7. | ***for*** item in *neig_list (index + 1):*
8. | | ***if*** *neig_list contains similar neighbors*
9. | | clique found
10. |
12. **end**
11. **end**

The second version of the algorithm terminates after finding the first maximal clique. The pseudocode of this algorithm is given below.

Table 14 Proposed Algorithm Version 2

Proposed Algorithm V-2

Input:

$G = (V, E)$: a graph

v_r : first node to start

Output:

All possible cliques

Begin:

$sorted\_list$: contains all the nodes from highest degree to lowest

$neig\_list$: contains the neighbors of v_r with neighbors

$clique\_list$: contains all possible cliques

1. ***for*** *k in sorted_list:*
2. | *neig_list initialization*
3. | ***for*** *node(v_r) in k:*
4. | | *neig_list = node(v_r) + neighbors of v_r*
5. | |
5. | | *first index = contains node with neighbors*

```
    end
    for item in neig_list (index + 1):
        if neig_list contains similar neighbors
            First maximal clique found terminate the program immediately
            break
    end
end
```

CHAPTER 5

5 RESULTS

This chapter mainly compares the results of the proposed algorithm with the other algorithms. The efficiency and performance of the proposed method are measured by applying it to different real-world datasets. This study consists of two algorithms. The first algorithm finds all possible cliques from the datasets and the second version one maximum clique. The performance of the proposed study (first version) is compared with the B.K. (Bron Kerbosch) algorithm and second The CLIQUES algorithms. Tables 15 and 16 illustrate a comparison in the perspective of performances between the proposed study and the Bron Kerbosch algorithm. The performance metrics consist of processing time between two algorithms.

As the proposed algorithm has two versions; the second version of the algorithm finds a maximum clique and terminates. This study does not compare the performance of the second version of the proposed method with the BK algorithm as the second version only finds a maximum clique from a given graph.

The performance time between the two algorithms is measured in milliseconds. The results are illustrated in the table below. The results show that the proposed algorithm performs better than the Bron Kerbosch algorithm. However, the CLIQUES algorithm is much faster than all others.

Table 15 Comparison of the methods for datasets of social networks

Processing Time (Milliseconds)	Graph Name (# nodes, #edges)		
	Government (2553, 8406)	**Media** (307, 237)	**Politician** (1250, 2036)
BK Algorithm	0.143917	0.133932	0.111940
CLIQUES Algorithm	0.00299	0.002000	0.001997
Proposed M. V-1	0.141927	0.016000	0.087685
Proposed M. V-2	0.072014	0.007977	0.023977

Table 16 compares the processing time between the proposed algorithm, the Bron-Kerbosch algorithm and the CLIQUES algorithm on different randomly generated graphs. Similarly, the results of the proposed methods applied on the random graphs show that the proposed method is much faster than the BK algorithm. However, CLIQUES algorithm performs similar to the proposed method for smaller random graphs and some of the larger graphs.

Table 16 Comparison of the methods for datasets of random graphs

Processing Time (Milliseconds)	Graph Name (# nodes, #edges)					
	Random Graph-1 (23, 41)	**Random Graph-2** (46, 83)	**Random Graph-3** (50, 600)	**Random Graph-4** (100, 800)	**Random Graph-5** (100, 1000)	**Random Graph-6** (100, 1200)
CLIQUES Algorithm	0.0019993	0.002998	0.001998	0.0010039	0.0019993	0.002997
Born Kerbosch-Algorithm	0.0049982	0.005974	0.04197	0.24921	0.0039963	0.004998
Proposed Method V1	0.0019950	0.002997	0.005012	0.023987	0.0159964	0.001995
Proposed Method V2	0.0010023	0.002000	0.00399	0.0079994	0.0040020	0.001002

CHAPTER 6

6 DISCUSSION AND FUTURE WORK

This chapter discusses the results of our proposed method that consists of two variants. The first version finds all possible cliques from social networks and presents a solution to questions such as "What are the relationships within a social network" and "How are people related to each other in a complex social network." The proposed algorithm is tested on multiple datasets as well as on random graphs. The algorithms perform better than the well-known the Bron Kerbosch method that is highly recognized in social networks to find all cliques and CLIQUES algorithm. Furthermore, the study proposes another efficient method to find a maximum clique.

In the future, the main focus is to implement the proposed method on larger scale real-world datasets. The future target is to increase the precision and robustness of the proposed algorithm. Moreover, this method can be applied to more dynamic networks. Furthermore, the time and space complexity can be reduced and can be compared with some other recently proposed other algorithms.

7 . REFERENCES

[1]. Aric A. Hagberg, D. A. (2008). Exploring Network Structure, Dynamics, and Function Using NetworkX. *Proceedings of the 7 Python in Science Conference (SciPy 2008)*.

[2]. B. Balasundaram, S. B. (2006). Clique Relaxations in Social Network Analysis The Maximum k- plex Problem. *https://www.researchgate.net/publication/228620797_Clique_Relaxations_in_Social_Network_Analysis_The_Maximum_k-Plex_Problem*.

[3]. Brian Dickinson, B. V. (2013). A genetic algorithm or identifying overlapping communities in social networks using an optimized search space. *Social Networking ,2013*, 2(04):193.

[4]. Clara Pizzuti. (2009). A multi-objective genetic algorithm for community detection in networks. *ICTAI'09. 21st International Conference on. IEEE, 2009.*, 379–386.

[5]. E. Tomita, a. T. (2007). An Efficient Branch and Bound Algorithm for Finding a Maximum Clique with Computational Experiments. *, Journal of Global Optimization*, 95-11195-111.

[6]. E. Tomita, Y. S. (2010). A Simple and Faster Branch-and-Bound Algorithm for Finding a Maximum Clique. *WALCOM: Algorithms and Computation*, 191-203.

[7]. E.E. Santos, L. P. (2006). An Effective Anytime Anywhere Parallel Approach for Centrality Measurement in Social Network Analysis. *IEEE Systems, Man, & Cybernetics Society, 2006*, 8-11.

[8]. Etsuji Tomita, A. T. (2006). The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science 363* , 28 – 42.

[9]. Feng Zhang, H. L. (2018). An Adaptive Breadth-First Search Algorithm on Integrated Architectures. *The Journal of Supercomputing · November 2018*.

[10]. H. C. Johnston. (1976). Cliques of a Graph--Variations on the Bron-Kerbosch Algorithm . *International Journal of Computer and Information Sciences, Vol. 5, No. 3, 1976* .

[11]. Hadhemi Boubaker, W. K. (2020). Improved Overlapping Community Detection in Networks based on Maximal Cliques Enumeration. *24th International Conference on Knowledge-Based and Intelligent Information & Engineering.* Sousse, Tunisie.

[12]. Jacob Fox, T. R. (2018). Finding Cliques in Social Networks: A New Distribution-Free. *SIAM Journal on Computing · April 2018*, 29.

[13]. Jonas Hasselberg, P. M. (1993). Test case generators and computational results for the maximum clique problem. *Journal of Global Optimization,* , 463–482.

[14]. Lancichinetti A., F. S. (2009). Detecting the overlapping and hierarchical community structure of complex networks. *New Journal of Physics.*

[15]. Li, H. (2017). A New Algorithm for Enumerating All Maximal Cliques . *3rd IEEE International Conference on Computer and Communication.* Beijing, China.

[16]. M.R. Garey and D.S. Johnson. (1979.). Computers and Intractability: A Guide to the Theory of NP- Completeness. *Freeman, San Francisco, 1979.* .

[17]. Maity, S. (2014). Detection of Overlapping Communities in Social network. *PhD thesis*.

[18]. Mohammad Hussein Alomari, D. O. (January, 2017). Finding Cliques In Simulated Social Networks Using Graph Coloring Technique. *Degree of Master of Computer Science in Middle East University January, 2017* , 1-84.

[19]. Narsingh Deo. (2016). *Graph Theory with Applications to Engineering & Computer Science.* Mineola, New York: DOVER PUBLICATIONS, INC.

[20]. P. Santi. (2015). A Elements of Graph Theory. *Topology Control in Wireless Ad Hoc and Sensor Networks* .

[21]. Pizzuti, C. (2008). Ga-net: A genetic algorithm for community detection in social networks. . *In PPSN Springer*, 1081–1090.

[22]. Ranjit Kumar, R. K. (2017). Fast Algorithm for Enumerating Maximal Cliques in Large Scale Network. *978-1-5386-4318-1/17 ©2017 IEEE*.

[23]. Rozemberczki, B. a. (2019). GEMSEC: Graph Embedding with Self Clustering. *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2019* (pp. 65-72). ACM.

[24]. S.G.Shirinivas, S. (2010). APPLICATIONS OF GRAPH THEORY IN COMPUTER SCIENCE AN OVERVIEW . *International Journal of Engineering Science and Technology*, 4610-4621.

[25]. Santos, L. P. (2008). An Anytime-Anywhere Approach for Maximal Clique Enumeration in Social Network Analysis . *2008 IEEE*.

[26]. Sheila Eka Putri, T. N. (2011). Implementation and Analysis of Depth-First Search (DFS) Algorithm for Finding The Longest Path. *Conference Paper · August 2011(DOI: 10.13140/2.1.2878.2721)*.

[27]. Xingyi Zhang, C. W.-F. (2017). A Fast Overlapping Community Detection Algorithm Based on Weak Cliques for Large-Scale Networks. *IEEE TRANSACTIONS ON COMPUTATIONAL SOCIAL SYSTEMS, VOL. 4, NO. 4*, 13.

[28]. Yanan Cai, C. S. (2011). A novel genetic algorithm for overlapping community detection. *Advanced Data Mining and Applications*, 97–108.

[29]. Yanyan Xu, J. C.-C. (2015). Distributed Maximal Clique Computation and Management. *IEEE TRANSACTIONS ON SERVICES COMPUTING, VOL. X, NO. X, JULY-SEPTEMBER 2015*.