Solving Maximum Clique Problem using Using Branch and Bound Method

Bikramjeet Singh
School of Computer Science
University of Windsor
singh3h2@uwindsor.ca

Harjot Singh Saggu
School of Computer Science
University of Windsor
saggu3@uwindsor.ca

Jaskaran Singh Luthra
School of Computer Science
University of Windsor
luthraj@uwindsor.ca

***Abstract*—The maximum clique problem is a notoriously difficult problem, along with other combinatorial optimization problems like graph coloring and the traveling salesman problem. There are several methods for solving the maximum clique problem, including brute force, backtracking, and branch and bound algorithms. In this paper, we focus on the bound and branch method for solving the maximum clique problem. We explore the theoretical background of this method and review the existing literature on its application to the maximum clique problem. We also provide a detailed description of the algorithm and present experimental results demonstrating its effectiveness compared to other state-of-the-art methods. Our results show that the bound and branch method is a promising approach for solving the maximum clique problem on a wide range of instances.**

***Index Terms*—Maximum Clique, branch and bound, graph coloring, heuristic, greedy**

I. Team Contribution

We divided the project in 3 different parts and we have mentioned the contribution briefly of each team member below:

Harjot Singh: Conducted research on the maximum clique problem in graph coloring and identified the limitations of using only the greedy algorithm. Proposed the use of branch and bound to facilitate maximum clique search and gathered relevant literature on the topic.

Jaskaran Singh: Developed the branch and bound algorithm for the maximum clique problem, building on the concepts of integer programming and graph theory. Conducted experiments to test the algorithm's effectiveness and accuracy, and provided detailed analysis of the results.

Bikramjeet Singh: Reviewed and edited the paper, ensuring that the writing was clear, concise, and consistent. Provided feedback and suggestions for improving the presentation of the research and the algorithm, and worked with A and B to address any issues or concerns that arose during the project.

II. Introduction

In this project we have discussed the branch and bound procedure to solve the maximum clique problem. In the discussion of the method described on the upper and lower bound by implementing a greedy heuristic and branching procedure for determining the maximum clique (G) and minimum coloration (G) in the graph G. Solution the maximum clique problem with this method is beneficial because it provides an alternative perspective on branch and bound, but it is also because they do not need to explain software optimization.

A description of the basic concepts of graph theory to a wider survey refer to [3], [4] and [5]. Since the branch and bound on pure integer programming area can seem abstract, use simple graph algorithm to describe the procedure can be made more visual and easy to understand. Hence, this project covers completion maximum clique using the branch and bound.

III. Graph Theory

Graph theory is a branch of mathematics that studies the properties of graphs, which are mathematical structures that model pairwise relationships between objects. A graph is a collection of vertices (also known as nodes) and edges, which connect pairs of vertices. In graph theory, these vertices and edges can be represented by various mathematical notations and symbols, such as a matrix or a set of equations [2].

The study of graph theory is an essential part of modern mathematics, as it provides a powerful framework for modeling complex systems and analyzing their properties. Graph theory has applications in a wide range of fields, including computer science, biology, physics, economics, and social sciences. Graph theory can be used to model a wide range of real-world systems, such as social networks, transportation networks, biological networks, communication networks, and financial networks.

One of the important concepts in graph theory is a clique. A clique in a graph is a subset of vertices that are all mutually adjacent. In other words, every vertex in the clique is directly connected to every other vertex in the clique. Cliques are an important concept in graph theory, as they provide a way of characterizing the "Density" of a graph. The maximum clique of a graph is the largest clique that can be found in the graph, and is an important measure of the graph's structure [8].

Cliques have many applications in various fields, such as social network analysis, biology, and computer science. For example, in social network analysis, cliques can represent groups of individuals who are tightly connected with each other. In biological networks, cliques can represent groups of genes that interact with each other. In computer science, cliques are used in the study of parallel algorithms and distributed computing.

One of the important graph theory is the maximum clique problem. The maximum clique problem is the problem of finding the largest clique in a given graph. This problem is known to be NP-hard, which means that it is computationally difficult to solve for large graphs. However, there are several algorithms that can solve the maximum clique problem for small and medium-sized graphs. These algorithms are based on various techniques, such as branch and bound, backtracking, and heuristic search [3].

IV. Literature Review

The purpose of the literature review section is to provide an overview of the existing research on solving the maximum clique problem using greedy approach and branch and bound methods. This section is divided into four subsections: Introduction, Review of Literature, Analysis and Comparison, and Conclusion. The Introduction defines the problem and explains its significance, while the Review of Literature summarizes the key ideas and findings of the selected sources. The Analysis and Comparison section compares and contrasts the various branch and bound algorithms used in the literature, discussing their strengths and limitations. Finally, the Conclusion summarizes the section's main points and highlights any gaps or limitations in the existing research, setting the stage for the subsequent sections of the paper.

The theoretical orientation of this study is based on the field of computational complexity theory, which seeks to classify and compare computational problems in terms of their inherent difficulty. The maximum clique problem is an NP-hard problem, meaning that there is no known algorithm that can solve it in polynomial time [4]. To address this challenge, researchers have developed various approximation algorithms and exact algorithms based on branch and bound methods.
To select literature for this review, we conducted a search of the IEEE Xplore, SpringerLink, and ScienceDirect databases using keywords such as "maximum clique problem," "branch and bound," and "exact algorithms." Few papers selected for this review are:Panos M. Pardalos, Gregory P. Rodgers,A branch and bound algorithm for the maximum clique problem,Computers and Operations Research, Volume 19,1992 ; Grosso, A., Locatelli, M. and Croce, F.D. Combining Swaps and Node Weights in an Adaptive Greedy Approach for the Maximum Clique Problem [2].

According to A. Grosso and F. Della Croce [2], the greedy algorithm can get trapped in local optima and fail to find the global optimum, especially for large and complex graphs. The authors argue that the performance of the greedy algorithm is limited by its inability to explore the search space beyond the immediate neighborhood of the current solution. Moreover, the greedy algorithm is often unable to exploit the structure and symmetry of the graph, leading to inefficient search and suboptimal solutions. To overcome these limitations, researchers have proposed more sophisticated algorithms based on branch and bound methods.
While the papers do not explicitly discuss the theoretical framework of computational complexity theory, they are relevant to this study because they provide practical insights into the implementation and optimization of branch and bound algorithms.

In the paper, C. Pardalos and V. Vazirani [1] propose an efficient algorithm for the maximum clique problem. The authors introduce a new graph representation and use it to develop a heuristic algorithm that combines elements of both the greedy and backtracking approaches. The algorithm is based on a modified version of the Bron-Kerbosch algorithm and uses various heuristics to improve its efficiency, such as degree ordering and maximum degree reduction. The authors compare their algorithm to other state-of-the-art algorithms and show that it outperforms them on a variety of benchmark instances. Furthermore, the authors provide a detailed complexity analysis of their algorithm and show that it has a worst-case time complexity of $O(2^{(n/2)})$, where n is the number of nodes in the graph. Overall, the paper makes a significant contribution to the field of maximum clique algorithms and provides a valuable addition to the literature on this topic.

V. Graph Colouring

A graph G = (V, E) is a set of vertices and edges. (Along this paper, the vertex will be represented by letters and colors for the vertices will be represented by integers). A clique in G is part of the vertices C $\subseteq$ V where each pair of vertices are joined with an edge, for example, see Figure 1. In other words, all the vertices in the clique must adjacent to one another. The size of the maximum clique in G indicated by (G). An illustration of a nice, simple, this problem is with a dinner party. Suppose the vertices of the graph represent the dinner guests. Suppose edge connects guests if they already know each other. Then the maximum clique will be the biggest set of guests at a dinner where everyone knew each other [3].

A vertices coloring is to assign a color to the vertices of a graph G such that no two adjacent vertices receive the same color. A minimum coloring is a coloring that uses the least amount possible for the entire graph coloring, as an example, see Figure 2. The size of the minimum coloring in

G indicated by (G). Minimum Coloring can also be illustrated using the example of a dinner party described above. Suppose that the host wants her friends to meet each other. Then it will just sit people at tables where they are not familiar with the other guests. If we let the different colors represent the different tables at the party, then the only guests at no edge connecting them (ie, guests who do not know each other) will be allowed to color / same table. Coloring minimum in accordance with the minimum number of tables required to ensure no guests sit with people they've known.

Consider a triangle, which is a clique of size 3. Each of the three vertices must have different colors, if not, then there will be no edge has endpoints with the same color and coloring will be valid. In general, in any coloring a graph G, each pair of adjacent vertices must have different colors. Therefore, (G) $\leq$ (G).

More importantly, every clique in G is a lower bound on (G), and every shade in G is an upper bound on ω (G). In the next section, given a heuristic for determining the clique and color and, hence, obtain the boundaries to be used in a branch and bound algorithm [9].

VI. Branch and Bound

The branch and bound algorithm has been widely applied to solve various combinatorial optimization problems [4, 10, 11]. The main objective of the branch and bound approach is to break down the given problem into a number of smaller partial sub problems. Each sub problem is further divided unless it can be proven that it cannot produce an optimal solution or it cannot be decomposed further. The order of testing or decomposing partial problems depends on the chosen search technique. The best-bound search, depth-first search, breadth-first search, and heuristic-based search techniques are some of these strategies [1].

The goal of branch and bound algorithms is to search the entire branch and bound tree for a global optimum. It might not be practical to search every branch and bound tree, though. As a result, many branch and bound implementations include clauses for halting execution after a set amount of time. Consequently, there are two main goals for branch and bound algorithms [4]. The first step is to restrict the search space so that all potential solutions are implicitly listed. The second step is to select the best option from the space of potential solutions.

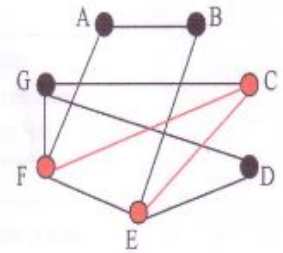

Figure 1. Vertices C, E, and F form clique size 3

VII. Procedure of Branch and Bound

A. Determine the Upper and Lower Bound

To find a lower bound (LB) on the size of the maximum clique, the search heuristic greedy clique can be implemented in a simple graph G = (V,) as follows [4]:

1. Suppose K = θ.
2. Suppose v is a vertices in G but not in K with the maximum degree. Adding v in K.

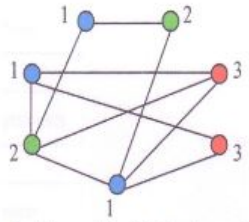

Figure 2. Shows the Coloring of Size 3

3. Remove all vertices that are not adjacent to v from V.
4. If V is empty, then stop. If not, then go back to step 2.

Basically, added to the clique K vertices with the greatest degree that is connected to all the vertices, and are already in the clique. The cardinality of the set K is generated is a clique in the graph G and therefore, lower bound on ω(G), is the size of the maximum clique in G [2, 7]. Figure 3 illustrates this heuristic.

For more details, look at the following discussion.

Step 1. Let K = . Determine the maximum vertices degree, the vertices F (Figure 3.a.);

Step 2. Adding F to K, so that sehingga K = F, then remove C is not adjacent with F (Figure 3.b.);

Step 3. Adding D to K, so that K = F, D, which is not clear adjacent E with D (Figure 3.c.);

Step 4. Adding B to K, so that K = F, D, B(Figure 3.d.);

Step 5. Adding A to K, so that K = F, D, B, A. In order to get the maximum clique, i.e, (G) $\geq$ 4 = lower bound (LB) (Figure 3.e.).

Figure 3. a, b, c, d, and e, is an illustration of the heuristic greedy clique, which (G) $\geq$ 4.

To find the upper bound (UB) on the size of the maximum clique, (heuristic greedy coloring) can be implemented in a simple graph G = (V, E) as follows [4]:

- Colourise vertices v1 with color c1.
- For each vertices vi in a row, selected the lowest numbered color that does not result in coloring is not valid (ie, such that no other vertices to vi adjacent previously colored with the same color). Figure 4 illustrates the heuristic coloring on the same graph as in Figure 3.

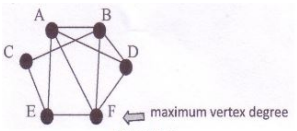

Figure 3.a.

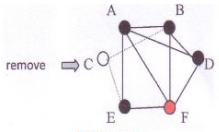

Figure 3.b.

Step 1. Select the maximum vertices degree, the vertices A given color 1 (red);
Step 2. Select the vertices that adjacent and degree less than or equal to A, that is vertices B are colored 2 (blue);
Step 3. Select the vertices that adjacent and degree less than or equal to A and B, the vertices F are color 3 (green);
Step 4. Select the vertices that adjacent with F and A, namely vertices E are colored 2 (blue);
Step 5. Select the vertices that adjacent with E and B, the vertices C are color 1 (red);
Step 6. Last vertices D are colored 4 (yellow), because adjacent with color 1 vertices A, vertices B adjacent with color 2, and adjacent with vertices F color 3.

Figure 4. An illustration Coloring Heuristic Greedy, where 4 $\leq \omega(G) \leq \omega(G) = 4$

The results of this heuristic is a valid coloring of G, which is the upper bound on (G), the size of the maximum clique in G. Once the basics of graph theory has been reached, then to solve the problem on any graph of maximum clique appropriately measured using the branch and bound.

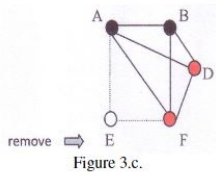

Figure 3.c.

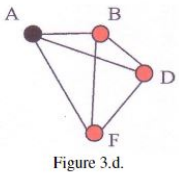

Figure 3.d.

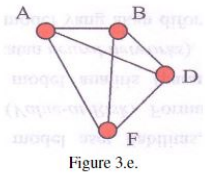

Figure 3.e.

B. Branching Procedure

In this section of the procedure branching, using Figure 5, which describes the the branching following procedure:

1. Given graph G = (V, E), selected branch vertices v $\in$ V, where v is any vertices not connected to all other vertices in G. If there is no such v, then G is a clique. Else next step.

2. Create a graph of the G' and G" of G as follows:

- G' is the graph part of G induced by the vertices V-v, i.e G' formed by deleting the vertices v (and adjacent edges) of G.

- G" is part of the graph G induced by vertices v in N(v), where N(v) indicates the neighbors of the vertices v, i.e. G" is formed by retaining only the vertices v and all vertices that adjacent to him in G.

Vertices maximum degree of a member or not a member of the maximum clique. Since removing G' only maximum vertices degree of a graph parent, and G" maintain the maximum degree vertices and all its neighbors, the way in which the sub graph G' and G" was made to ensure that a maximum clique in G will still be there in one or both of the sub graph. For more details, see [3, 4]. Thus, the maximum clique is not destroyed in this process, but the size of the graph at each the branching vertices decreases with each iteration.

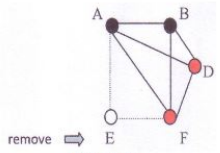

Figure 3.c.

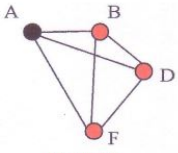

Figure 3.d.

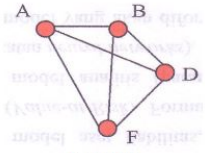

Figure 3.e.

VIII. Calculation Result of Branch and Bound

Now that the branching procedure and calculation boundary has been explained, the entire branch and bound algorithm can be further illustrated. Figure 6 shows the sequence of branching processes vertices, using upper and lower bounds to prune the branches, and the clique and coloring searched using heuristic described above. Clique is outlined in black, and coloring's are shown generated using heuristic greedy clique. Clique alternatives can be sought when different options to the maximum degree of a node is made (in the case of ties), and color [1]. alternatives may be sought if the order of vertices different from the one shown above.
Step 1. Clique = F,G,I; LB = 3 and Color = UB = 4 Branching on the vertices G
Step 2. Remove vertices G, obtained Clique = A, E, F, I, LB = 4. Color = UB = 4
Step 3. Remove all vertices that are not adjasen with G, obtained Clique = F,G,I; LB = 3. Color = UB = 3.
Step 4. Remove vertices F in Step 2, obtained Clique = A, E, I; LB = 3. Color = UB = 4
Step 5. Remove all vertices that are not adjacent with vertices F in step 2, obtained Clique = A, E, F, I; LB = 4. Color = UB = 4.

Figure 6, from step 1 to step 5, is an example of the whole procedure using branch and bound method in solving the maximum clique problem.

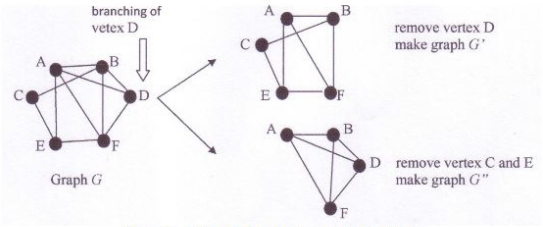

Figure 5. An Illustration of Branching Procedure

IX. Result

The table shows the results of finding cliques for various graphs using the Branch and Bound method. In this method, an initial clique is formed and then extended by adding vertices one by one. At each step, the algorithm checks if the current partial clique can be extended further or not. If not, it backtracks to the previous step and tries to extend the clique in a different way.

The first column of the table indicates the name of the graph file. The second column shows the size of the largest clique found in that graph using the Branch and Bound method. The third column indicates the number of vertices in the graph, and the fourth column represents the number of edges in the graph.

The last column shows the time taken in milliseconds to find the largest clique for that graph using the Branch and Bound method. The time taken to find cliques varies greatly depending on the size and complexity of the graph. For example, the algorithm was able to find the largest clique for the '4' vertex graph in only 0.030 ms, while it took 20132.100 ms to find the largest clique for the 'johnson8-4-4.clq.txt' graph.

Overall, the Branch and Bound method is an effective way to find cliques in graphs, but its time complexity can be quite high for larger graphs.

	filename	nodes	edges	clique lengt	time
0	sample.col	4	5	3	0.030 ms
1	huck.col	74	301	11	11.177 ms
2	miles250.col	125	387	8	0.234 ms
3	anna.col	138	493	11	64.904 ms
4	homer.col	556	1629	13	192.321 ms
5	le450_5a.col	450	5714	5	1703.736 ms
6	le450_5b.col	450	5734	5	1790.787 ms
7	le450_15b.col	450	8169	15	1036.543 ms
8	le450_25a.col	450	8260	25	54.331 ms
9	le450_5d.col	450	9757	5	2814.926 ms
10	le450_5c.col	450	9803	5	2788.681 ms
11	le450_15c.col	450	16680	15	6621.513 ms
12	le450_15d.col	450	16750	15	6650.191 ms
13	le450_25c.col	450	17343	25	6442.673 ms
14	le450_25d.col	450	17425	25	5336.674 ms
15	johnson8-2-4.clq.txt	28	210	4	9.170 ms
16	hamming6-4.clq.txt	64	704	4	55.660 ms
17	c-fat200-1.clq.txt	200	1534	12	226.391 ms
18	hamming6-2.clq.txt	64	1824	32	0.509 ms
19	johnson8-4-4.clq.txt	70	1855	14	20132.100 ms
20	c-fat200-2.clq.txt	200	3235	24	0.837 ms
21	c-fat500-1.clq.txt	500	4459	14	1.275 ms
22	c-fat200-5.clq.txt	200	8473	58	1110.305 ms
23	c-fat500-2.clq.txt	500	9139	26	2.022 ms
24	p_hat300-1.clq.txt	300	10933	8	11720.020 ms
25	c-fat500-5.clq.txt	500	23191	64	5.271 ms
26	hamming8-2.clq.txt	256	31616	128	16.735 ms
27	c-fat500-10.clq.txt	500	46627	126	16.667 ms
28	hamming10-2.clq.txt	1024	518656	512	987.168 ms

X. Conclusion

Branch and bound method has been used to the search the maximum clique problem. To determine the chromatic number (G)of a graph G has been sought by the greedy heuristic, chromatic number (G)is the upper bound (UB) on G and by branching procedure obtained lower bound (LB) is the maximum clique (G) on G. Results of the maximum clique the search also shows that the maximum clique (G) is less than or equal premises chromatic number (G). The use of branch and bound method is advantageous, because it provides an alternative perspective to solve the maximum clique problem, as well as the optimal method of solution requires no software optimization.

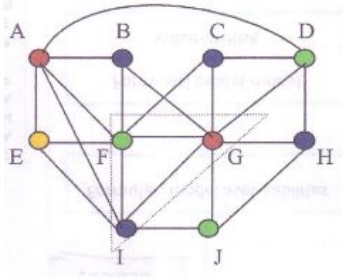

Figure 6. The sequence of branching processes vertices

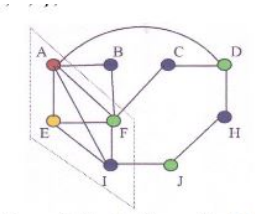

Figure 6. Step 2, Clique={A,E,F,I}, LB=4. Color=UB=4

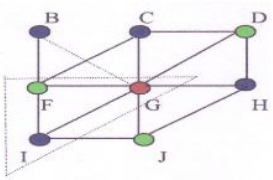

Figure 6. Step 3, Clique={F,G,I}, LB=3. Color=UB=4

Acknowledgment

We, the team, would like to express our sincere gratitude to our professor, Dr. Luis Reuda, for his unwavering support, guidance, and mentorship throughout the project. We also want to thank our colleagues for their support and feedback, and our institution for providing the resources and facilities necessary for conducting the research. Lastly, we are deeply grateful to our families and friends for their encouragement and belief in us. We appreciate the opportunity to learn from Dr. Luis Reuda and the invaluable experience we have gained from working with him.

References

[1] Panos M. Pardalos, Gregory P. Rodgers,A branch and bound algorithm for the maximum clique problem,Computers and Operations Research, Volume 19, Issue 5,1992,ISSN 0305-0548.

[2] Grosso, A., Locatelli, M. and Croce, F.D. Combining Swaps and Node Weights in an Adaptive Greedy Approach for the Maximum Clique Problem. Journal of Heuristics 10, 135–152 (2004).

[3] . Bran and J. Kerboscb,Finding all cliques of an undirected graph Coals. Ass. Come. Mach. 16,575-577(1973).
[4] Suyudi, M. Mohd, Ismail and Mamat, Mustafa Sopiyan, S. Supriatna, Asep. (2014). Solution of maximum clique problem by using branch and bound method. Applied Mathematical Sciences. 8. 81-90. 10.12988/ams.2014.310601.
[5] Bomze, I. M., M. Budinich, P. M. Pardalos, M. Pelillo. 1999. The maximum clique problem. D.- Z. Du, P. M. Pardalos, eds. Handbookof Combinatorial Optimization, Vol. 4. Kluwer, Boston, 1–74.
[6] Ostergard, P.R.J. 2002. A Fast Algorithm for the Maximum Clique Problem. Journal of Discrete Applied Mathematics, 120 (2002) 197-207.
[7] Pardalos, P. M., J. Xue. 1994. The maximum clique problem. J. Global Optim. 4(3) 301–328.
[8] Wilson, R. 1996. Introduction to Graph Theory. Pearson Education, Harlow, England.
[9] L. G. Mitten, Branch and bound method: general formulation and properties. Ops Res. IS, 24-34 (1970)
[10] E. Balas and C. S. Yu, Finding the maximum clique in an arbitrary graph. SIAM JI Cornput. f5, 0 -! 8 (1986).
[11] L. Gerhards and W. Lindenberg, Clique detection for nondirected graphs: two new algorithms. Comparing 21,295-322 (1979)
[12] E. L. Lawler and D. E. Wood, Branch and bound method: a survey. Ops Res. 14,699-719 (1969).
[13] T. Ibaraki, Enumerative approaches to combinatorial optimization. Arm. Ops Res. lO/ll (1987).
[14] P. M. Pardalos and G. Rodgers, Parallel branch and bound algorithms for quadratic zero-one programs on a hypercube architecture. Ann. Ops Res. 22, 271-292 (1990).

APPENDIX A: GRAPH FILE DATASET

Below are the Graphs constructed using the different files as part of input.

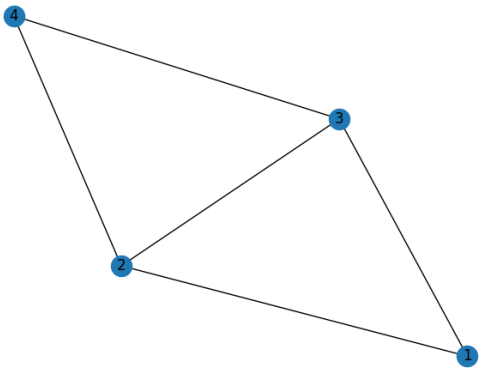

Figure 10. Sample Graph File

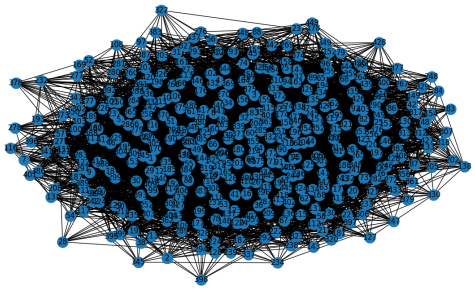

Figure 7. Le450 Graph File

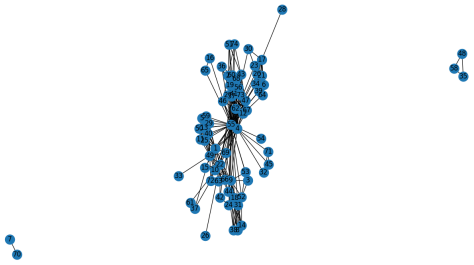

Figure 8. Huck Graph File

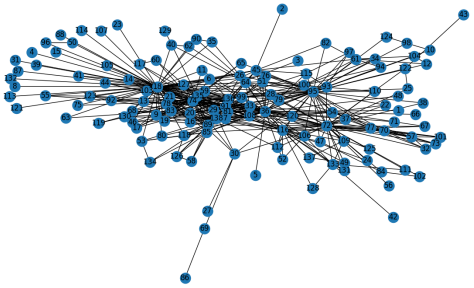

Figure 9. Anna Graph File

APPENDIX B: CODE SNIPPET

```
'''
The branch_bound_maximum_clique function is the main function that implements the branch and bound algorithm.
It uses the Bron-Kerbosch algorithm and the greedy clique and graph coloring heuristics to find the maximum clique in the graph.
The function starts with the graph, removes nodes that are not in a maximal clique, and creates two subgraphs.
Then, it recursively applies the procedure to each subgraph until it finds the maximum clique.
The function returns the maximum clique and the chromatic number of the graph.
'''
def branch_bound_maximum_clique(graph):
    max_clique = greedy_clique_heuristic_method(graph)
    chromatic_number = greedy_coloring_heuristic_method(graph)
    if len(max_clique) == chromatic_number:
        return max_clique
    else:
        g1, g2 = branching_method(graph, len(max_clique))
        return max(branch_bound_maximum_clique(g1), branch_bound_maximum_clique(g2), key=lambda x: len(x))
```

Fig. Funtion of branch and bound algorithm for finding the maximum clique in a graph.

The $bb_maximum_clique()$ function implements the branch and bound algorithm for finding the maximum clique in a graph. It uses the Bron-Kerbosch algorithm to find all maximal cliques, and prunes the search tree using the lower and upper bounds obtained from the greedy clique and coloring heuristics. The function returns the maximum clique found.

```
# The greedy_clique_heuristic_method function is a greedy search for cliques by iterating through nodes
# with the highest degree and filtering only their neighbors.
def greedy_clique_heuristic_method(graph):
    '''
    Greedy search for clique iterating by nodes
    with highest degree and filter only neighbors
    '''
    K = set()
    nodes = [node[0] for node in sorted(nx.degree(graph),
                                        key=lambda x: x[1], reverse=True)]
    while len(nodes) != 0:
        neigh = list(graph.neighbors(nodes[0]))
        K.add(nodes[0])
        nodes.remove(nodes[0])
        nodes = list(filter(lambda x: x in neigh, nodes))
    return K
```

Fig. Output

The $greedy_coloring_heuristic()$ function implements a greedy graph coloring algorithm with a degree order rule. It returns the number of colors used to color the graph. This function is used to find an upper bound on the maximum clique size.

```
# The bronk_algorithm function is an implementation of the Bron-Kerbosch algorithm for finding all maximal cliques in a graph.
def bronk_algorithm(graph, P, R=set(), X=set()):
    '''
    Implementation of Bron-Kerbosch algorithm for finding all maximal cliques in graph
    '''
    if not any((P, X)):
        yield R
    for node in P.copy():
        for r in bronk_algorithm(graph, P.intersection(graph.neighbors(node)),
                                 R=R.union(node), X=X.intersection(graph.neighbors(node))):
            yield r
        P.remove(node)
        X.add(node)
```

Fig. Output

The bronk() function implements the Bron-Kerbosch algorithm, a recursive algorithm for finding all maximal cliques in a graph. It takes in a graph, a set P of potential nodes to add to the current clique, a set R of nodes already in the current clique, and a set X of nodes already excluded from the current clique. It returns a generator object that generates all maximal cliques in the graph.

```
# The greedy_coloring_heuristic_method function is a greedy graph coloring heuristic with the degree order rule.
def greedy_coloring_heuristic_method(graph):
    '''
    Greedy graph coloring heuristic with degree order rule
    '''
    color_num = iter(range(0, len(graph)))
    color_map = {}
    used_colors = set()
    nodes = [node[0] for node in sorted(nx.degree(graph),
                                        key=lambda x: x[1], reverse=True)]
    color_map[nodes.pop(0)] = next(color_num)  # color node with color code
    used_colors = {i for i in color_map.values()}
    while len(nodes) != 0:
        node = nodes.pop(0)
        neighbors_colors = {color_map[neighbor] for neighbor in
                            list(filter(lambda x: x in color_map, graph.neighbors(node)))}
        if len(neighbors_colors) == len(used_colors):
            color = next(color_num)
            used_colors.add(color)
            color_map[node] = color
        else:
            color_map[node] = next(iter(used_colors - neighbors_colors))
    return len(used_colors)
```

The $greedy_coloring_heuristic()$ function implements a greedy graph coloring algorithm with a degree order rule. It returns the number of colors used to color the graph. This function is used to find an upper bound on the maximum clique size.

```
# The branching_method function is the branching method procedure.
def branching_method(graph, cur_max_clique_len):
    '''
    branching_method procedure
    '''
    g1, g2 = graph.copy(), graph.copy()
    max_node_degree = len(graph) - 1
    nodes_by_degree = [node for node in sorted(nx.degree(graph),  # All graph nodes sorted by degree (node, degree)
                                               key=lambda x: x[1], reverse=True)]
    # Nodes with (current clique size < degree < max possible degree)
    partial_connected_nodes = list(filter(
        lambda x: x[1] != max_node_degree and x[1] <= max_node_degree, nodes_by_degree))
    # graph without partial connected node with highest degree
    g1.remove_node(partial_connected_nodes[0][0])
    # graph without nodes which is not connected with partial connected node with highest degree
    g2.remove_nodes_from(
        graph.nodes() -
        graph.neighbors(
            partial_connected_nodes[0][0]) - {partial_connected_nodes[0][0]}
    )
    return g1, g2
```

The branching() function takes in a graph and the current maximum clique size found so far, and returns two smaller graphs obtained by removing nodes from the original graph. The removed nodes are chosen according to a branching strategy that is based on the node degrees.

```
You, 1 second ago | 2 authors (Jaskaran Singh and others)
import networkx as nx
import matplotlib.pyplot as plt

# read the input file
filename = './samples/anna.col'
with open(filename, 'r') as f:
    lines = f.readlines()

# parse the input file
G = nx.Graph()
for line in lines:
    if line.startswith('e'):      Jaskaran Singh,
        nodes = line.split()[1:]
        G.add_edge(int(nodes[0]), int(nodes[1]))

# plot the graph
nx.draw(G, with_labels=True)
plt.show()
```

In this function we are taking the file (graph file in our case) as input and parsing the graph file to eventually construct/plot the graph on which we're finding the clique and calculating the time taken to detect and find clique.