FINDING A MAXIMUM CLIQUE OF A CHORDAL GRAPH BY REMOVING VERTICES OF MINIMUM DEGREE

A thesis submitted

to Kent State University in partial

fulfillment of the requirements for the

degree of Master of Science

by

Sudipta Bhaduri

May 2008

Thesis written by

Sudipta Bhaduri

B.E., North Bengal University, India, 2003

M.S., Kent State University, USA 2008

Approved by

_Dr. Feodor F. Dragan_____, Advisor

_Dr. Robert A. Walker_____, Chair, Department of Computer Science

_Dr. Jerry Feezel_____ , Dean, College of Arts and Science

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

I wish to express my thanks to Professor Feodor F. Dragan whose kind help made this paper possible

DEDICATION

I wish to dedicate this paper to my father late. Nani Gopal Bhaduri

CHAPTER 1

Introduction

A "***graph***" refers to a collection of *vertices* and a collection of *edges* that connect pairs of vertices. A graph or *undirected* graph is an *ordered pair* G: = (V,E) where V is a set whose elements are called *vertices* and E is a set of pairs of (unordered) distinct *vertices*, called *edges* or *lines*. A ***clique*** of a graph is its maximal *complete subgraph*. There always exits an edge between any two vertices in a *clique* of a graph.

One of the first classes of graphs to be recognized as being perfect was the class of triangulated graphs. An undirected graph G is called triangulated if every cycle of length strictly greater than 3 possesses a chord that is an edge joining two nonconsecutive vertices of the cycle. Triangulated graphs have also been *called chordal, rigid-circuit, monotone transitive* and *perfect elimination graphs.* A vertex x of G is called simplicial if its adjacency set $Adj(x)$ induces a complete subgraph of G, i.e., $Adj(x)$ is a clique (not necessarily maximum). Dirac [1961], and later Lekkerkerker and Boland [1962], proved that a triangulated graph always has a simplicial vertex (in fact at least two of them), and using this fact Fulkerson and Gross [1965] suggested an iterative procedure to recognize triangulated graphs based on this and the hereditary property. *Namely, repeatedly locate a simplicial vertex and eliminate it from the graph, until either no vertices remain or the graph is triangulated or at some stage no simplicial vertex exists and the graph is not triangulated*.

Let G = (V, E) be an undirected graph and let $\sigma = [v_1, v_2, ..., v_n]$ be an ordering of the vertices. We say that σ is a *perfect vertex elimination* scheme if each v_i is a simplicial vertex of the induced subgraph $G_{\{v_{1i}, ..., v_n\}}$. In other words, each set $X_i = \{v_j \in Adj(v_i) \mid j > i\}$ is complete. The lemma that every triangulated graph G= (V, E) has a simplicial vertex and if G is not a clique, then it has two nonadjacent simplicial vertices has been proved in the book mentioned above. From this lemma ,Fulkerson-Gross recognition procedure affords us a choice of at least two vertices for each position in constructing a perfect scheme for a triangulated graph. Therefore, we can freely choose a vertex v_n to avoid during the whole process, saving it for the last position in a scheme. Similarly, we can pick any vertex v_{n-1} adjacent to v_n to save for the (n-1) st position. If we continued in this manner, we would be constructing a scheme backward. This is exactly what Luker [1974] and Rose and Tarjan [1976] have done in order to give a linear-time algorithm for recognizing triangulated graphs. The version presented in Rose, Tarjan and Luker [1976] uses a lexicographic breadth-first search in which the usual queue of vertices is replaced by a queue of (unordered) subsets of the vertices which is sometimes refined but never reordered. The complexity of recognizing the triangulated graphs using this approach is O (|V| + |E|). Fast algorithms for the coloring, clique, stable set and clique-cover problems have been discussed in the book. Let G = (V, E) be a triangulated graph, and let σ be a perfect elimination scheme for G. It was first pointed out by Fulkerson and Gross that every maximum clique was of the form $\{v\} \cup X_v$ where $X_v = \{x \in Adj(v) \mid \sigma^{-1}(v) < \sigma^{-1}(x)\}$.

By the definition of σ , each $\{v\} \cup X_v$ is complete. Let w be the first vertex in σ contained in an arbitrary maximal clique A: then $A = \{w\} \cup X_w$. A triangulated graph on n vertices has at most n maximal cliques, with equality if and only if the graph has no edges. The algorithm for the chromatic number and maximum cliques of a triangulated graph has also been given in that book using a different approach. The input of the algorithm is the adjacency sets of a triangulated graph G and a perfect elimination scheme σ and the output is all maximal cliques of G and the chromatic number $x(G)$. The time complexity of the algorithm is O (|V| + |E|).

In this paper we discuss an algorithm for finding a maximum clique of a chordal graph by removing the vertices of minimum degree. We also show how to color the graph with minimum number of colors. One can find a maximum clique of a chordal graph by *perfect elimination ordering.* But it is a complicated algorithm. While the problem is *NP-complete* on general graphs, we here give a linear time algorithm to find a maximum clique of a chordal graph and also we can show that the size of a maximum clique of a chordal graph equals the chromatic number of the graph, i.e., we can color the whole graph with the number of colors equal the size of a maximum clique.

CHAPTER 2

Minimum Degree ordering

2.1 Preliminaries

Fact (Known) [2]: Any chordal graph which is not a clique has at least two nonadjacent simplicial vertices.

Let us denote the size of the maximum clique by $\varphi(G)$.

Lemma 1: Degree of any simplicial vertex in a chordal graph is at most $\varphi(G)$ - 1.

Proof: From the definition of simplicial vertex we know that its neighborhood should be a clique. Now if its neighborhood is a maximum clique whose size is $\varphi(G)$ then that simplicial vertex will have degree $\varphi(G)$-1 and it is the maximum possible degree of any simplicial vertex. Hence the proof.

Minimum Degree Ordering: Consider any ordering $(v_1, v_2, v_3 ....., v_i, ..., v_n)$ of vertices of a graph G. Let G_i be the induced subgraph of the graph G, $G_i = G(\{v_i, v_{i+1} ...., v_n\})$. Also let $d_i(v_j)$ be the degree of v_j vertex in G_i. We say that a ordering $(v_1, v_2, v_3 ....., v_i, ..., v_n)$ of vertices of G is a *minimum degree ordering* if for any i from 1 to n, vertex v_i is a vertex of G_i with minimum degree $d_i(v_i)$.

Lemma 2: Let $(v_1, v_2, v_3 ....., v_i, ..., v_n)$ be a minimum degree ordering of a chordal graph G. Then each vertex v_i has less than $\varphi(G)$ neighbors with larger indices.

Proof: By contradiction, we assume that a vertex v_i has $\geq \varphi(G)$ neighbors with larger index. We take v_i with those neighbors and form a set S. S is not a clique. Because if S is a clique then we will have a clique with size $\geq \varphi(G) + 1$, which is not possible as we already defined the size of the largest clique in the graph G as $\varphi(G)$.

Now from the Fact (Known) any chordal graph which is not a clique has two nonadjacent simplicial vertices. We can see that $(v_i, ..., v_n)$ is not a clique and it forms a chordal graph. Let us take one of the simplicial vertices v_x ,which is anywhere between v_i and v_n and $v_x \neq v_n$. From Lemma 1, the degree of the vertex v_x is at most $\varphi(G)$-1 which is less than the degree of the vertex v_i . Hence v_x should come before the vertex v_i in the minimum degree list. A contradiction arises with v_x coming later than v_i.

Thus our assumption was wrong, and hence the proof.

Lemma 3 Let $(v_1, v_2, v_3 ....., v_i, ..., v_n)$ be a minimum degree ordering of a chordal graph G. Let $\varphi(G)$ be a size of the maximum clique of G. Then the last $\varphi(G)$ vertices in minimum degree ordering form a maximum clique of G.

Proof: We consider vertex v_k with largest index such that $\deg_k(v_k) = \varphi(G) - 1$.

We have to show that vertices ($v_k, v_{k+1}.....v_n$) form a clique of size $\varphi(G)$ in G.

By contradiction, we assume that $v_k, v_{k+1}.....v_n$ is not a clique.

If $v_k, v_{k+1}.....v_n$ is not a clique we must have at least two nonadjacent simplicial vertices according to the Fact (Known) above. Let us take a vertex v_{k+j} in $v_k, v_{k+1}.....v_n$ which is a simplicial vertex. Now since v_{k+j} is a simplicial vertex it must have degree at most $\varphi(G)$ - 1. We considered v_k being the vertex with largest index with degree $\varphi(G)$ -1. But here we get a vertex v_{k+j} which have greater index than v_k and have degree at most $\varphi(G)$ -1. Hence v_{k+j} should come in place of v_k, and our assumption was wrong.

Thus, $v_k, v_{k+1}.....v_n$ is a clique and its size is $\varphi(G)$, as we took the degree of the vertex v_k in $v_k, v_{k+1}.....v_n$ to be $\varphi(G)$ - 1. Therefore the last $\varphi(G)$ vertices of a minimum degree ordering form a maximum clique, and the proof is completed.

Corollary 1: Let v_i be the rightmost vertex with largest degree (which is $\varphi(G)$ -1) in a minimum degree ordering of a chordal graph G. Then $v_i, v_{i+1}, ..., v_n$ form a maximum clique of G.

2.2 A O (|V| + | E |) algorithm to find a minimum degree ordering.

The algorithm to find a minimum degree ordering of a graph G is given below.

1. Represent graph G in the form of an extended adjacency structure, where the adjacencies of each vertex and its corresponding address are stored in a sequential list.
2. Find the degrees of the vertices of the graph G from the adjacency list.
3. Take the minimum degree vertex
4. Put it next in order.
5. Remove that vertex from the graph G.
6. Update degrees of the remaining vertices of the graph G.
7. Repeat 3-6 until no vertex remains in the graph G.

We can see from the algorithm above that it produces a minimum degree ordering of vertices. Now we need to use appropriate data structures and procedures to show that its complexity bound is O (|E| + |V|).

For each vertex we create a node in memory and use extended adjacency list to represent the graph, which stores also the addresses of the nodes representing the vertices.

To represent the graph as an adjacency list we have to read the input which is done in 2m (m is any integer representing the number of edges) time as we read each edge twice. Also when we create a node we can get the address of that node in constant time. So the line 1 in the above algorithm takes linear time.

When we find the degree of each vertex we consider each edge twice and thus it also takes 2m linear time. So the lines 1 and 2 have time complexity O ($\sum_j$ deg (v_j)), where v_j is any vertex of the graph.

We shall construct a degree structure composed of a doubly linked list of vertices of degree i for each i, with an array of headers pointing to the i-degree lists. This structure provides a “bucket sort” of the vertices by degree. We actually create basket for each

degree i which is doubly linked list and is empty at the beginning. We distribute the vertices into the baskets using the "bucket sort". We keep in some parameter say t the earliest double linked list which is not empty. At first extract the first vertex from that doubly linked list which takes O (1) time as we can directly access that vertex by the pointer of the particular vertex from the extended adjacency list.

Putting the vertices next in order takes O (1) time. We take a order list to put the vertices in order.

Now we have to change the degrees of the vertices adjacent to the minimum degree vertex we deleted before. We extract those vertices from their i-degree list (i may differ for each of the adjacent vertex) and insert them to the (i-1) –degree list corresponding to its new degree (i-1) in the current graph. We can directly access those vertices in constant time using their pointers from the extended adjacency list and also can insert them in the new list in constant time. We keep an array or rather a matrix of pointers which points to each i-degree list at the beginning of the list and at the end of the list so that we can insert in linear time in the new list.

The Minimum Degree Ordering Algorithm will not alter the adjacency structure but the degree structure will be updated with every vertex deletion to provide the degree structure for the current graph.

Now since $\sum_{J=1}^{|V|} \deg(v_j \mid G) = 2 \mid E \mid$, it follows from the implementation of the algorithm that it takes time O (|E| + |V|). This implementation requires only O(|V|) space in addition to the space for the adjacency and degree structures, the total space requirement is 2| E | + O(|V|) .

2.3 Examples

Let us explain the algorithm above with an example below. Let us take the graph G below with vertices a, b, c, d, e, and f.

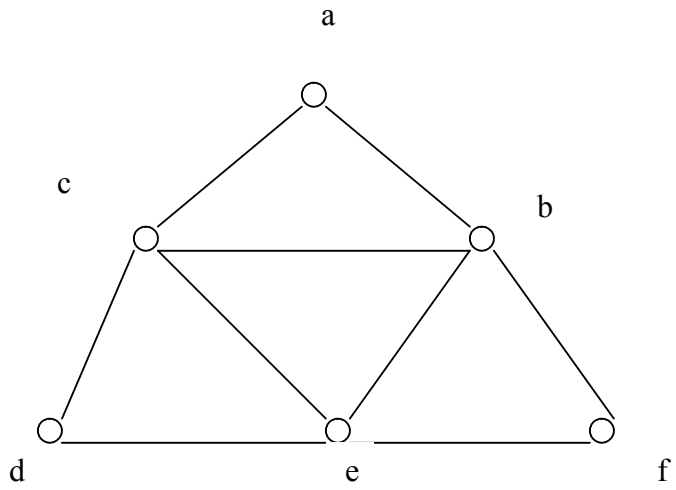

Figure 2.1. Graph G. Chapter 2 figure 1(of chapter 2)

To find a minimum degree ordering of the above graph G at first

we represent the graph as extended adjacency list.

In the adjacency list we list the neighbors of each vertex and also the corresponding pointers pointing to those vertices. We start from a as below.

$$a:\begin{bmatrix} b & c \\ p(b) & p(c) \end{bmatrix} \quad b:\begin{bmatrix} a & c & e & f \\ p(a) & p(c) & p(e) & p(f) \end{bmatrix} \quad c:\begin{bmatrix} a & b & e & d \\ p(a) & p(b) & p(e) & p(d) \end{bmatrix}$$

$$d:\begin{bmatrix} c & e \\ p(c) & p(e) \end{bmatrix} \quad e:\begin{bmatrix} d & c & b & f \\ p(d) & p(c) & p(b) & p(f) \end{bmatrix} \quad f:\begin{bmatrix} b & e \\ p(b) & p(e) \end{bmatrix}$$

Now according to the data structure we create a basket for each degree i which is doubly linked list and is empty at the beginning. Now when we put the above vertices in the linked lists it looks like below.

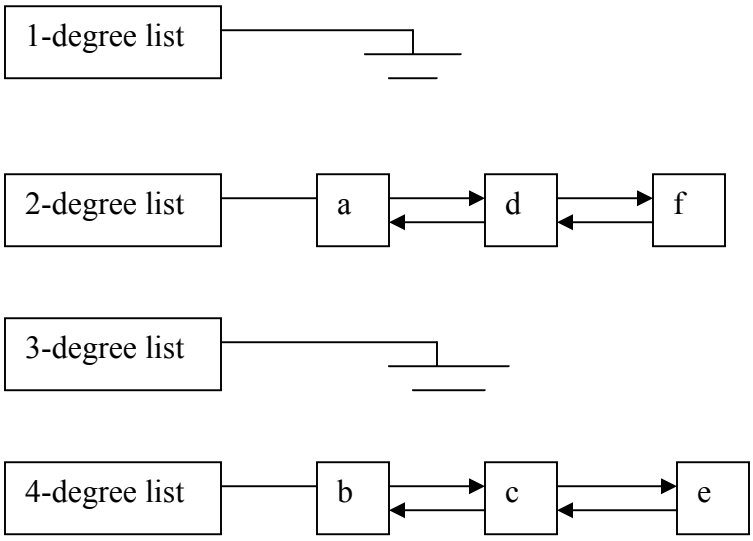

Figure 2.2. Initial Degree List. Chapter 2 figure 2(of chapter 2)

We keep in some parameter t the earliest double link list which is not empty, here t=2. At first we extract a and put it in the order list.

Now we have to change or update the degrees of b and c adjacent to a. We can directly go to b and c by their respective pointers $p(b)$, $p(c)$ and can extract them from their 4-degree list and then, by the use of the two pointers (front and rear pointer) stored in the

array we had taken before, we can insert them in the 3-degree list. After this the degree structure is

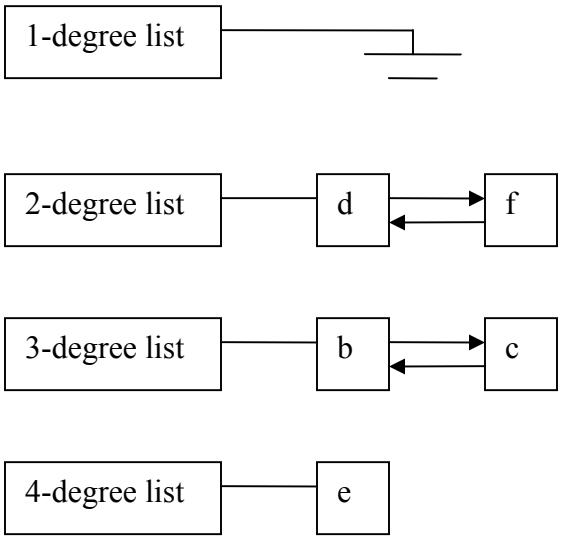

Figure 2.3. Degree List after vertex deletion. Chapter 2 figure 3(of chapter 2)

And the graph looks as below.

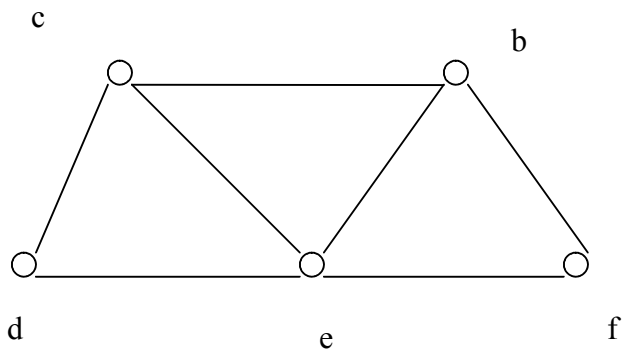

Figure 2.4. Graph G after vertex deletion. Chapter 2 figure 4 (of chapter 2)

The value of t is still 2 as this doubly linked list still contains the minimum degree vertices of degree 2. Therefore next we extract d from this list and insert it in the ordered list. Hence, the ordered list now contains a, d. Now we have to change the degree of c and e adjacent to d. As we did before we can directly access c and e by their respective pointers p(c) and p (e) and insert them in the 2-degree list and 3-degree list respectively. So now the degree structure looks like

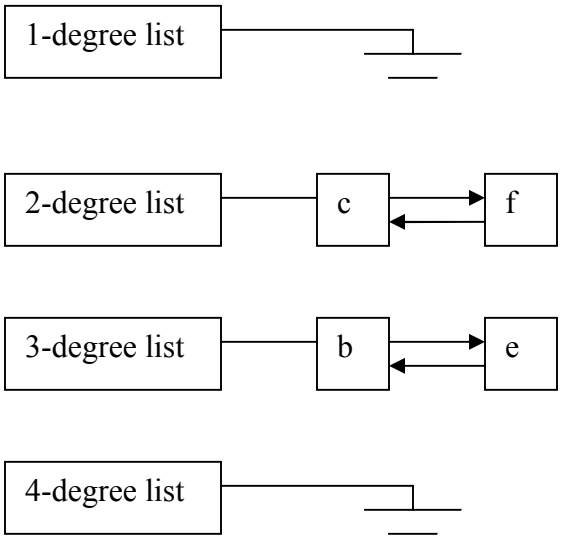

Figure 2.5. Degree List after vertex deletion. Chapter 2 figure 5 (of chapter 2)

And the graph looks like.

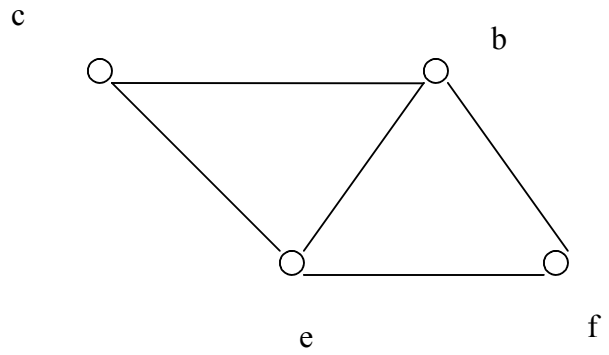

Figure 2.6. Graph G after vertex deletion. Chapter 2 figure 6 (of chapter 2)

Still $t = 2$. We extract c from this doubly linked list and insert it in the ordered list. Hence, the ordered list now contains a, d, c. We change the degrees of e and b adjacent to c. So the degree of e and b now becomes 2 and the degree list looks as below.

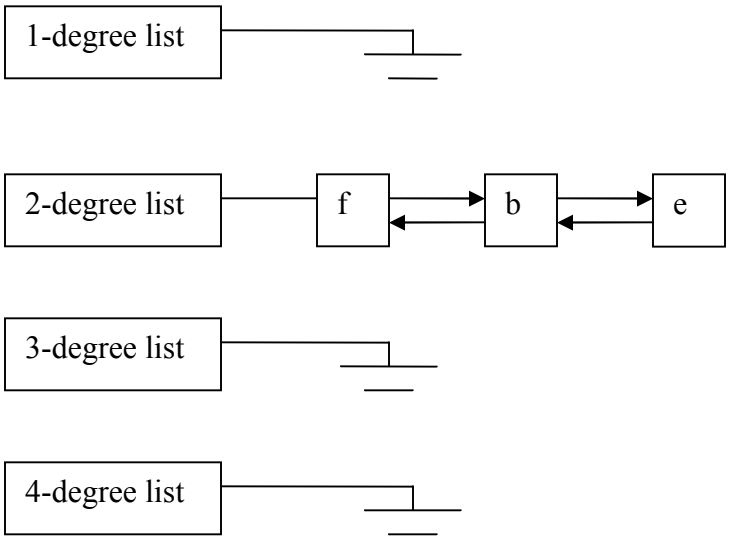

Figure 2.7. Degree List after vertex deletion. Chapter 2 figure 7 (of chapter 2)

The graph looks as below.

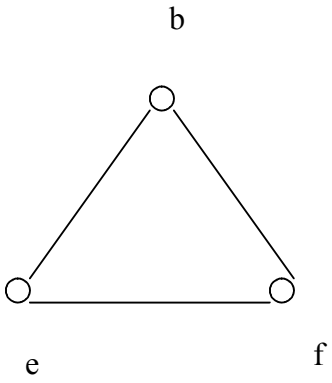

Figure 2.8. Graph G after vertex deletion. Chapter 2 figure 8 (of chapter 2)

Now $t = 2$. We extract f from the list and change the degrees of e and b to 1. The order list is a, d, c, and f. Hence the degree structure and the graph look as below.

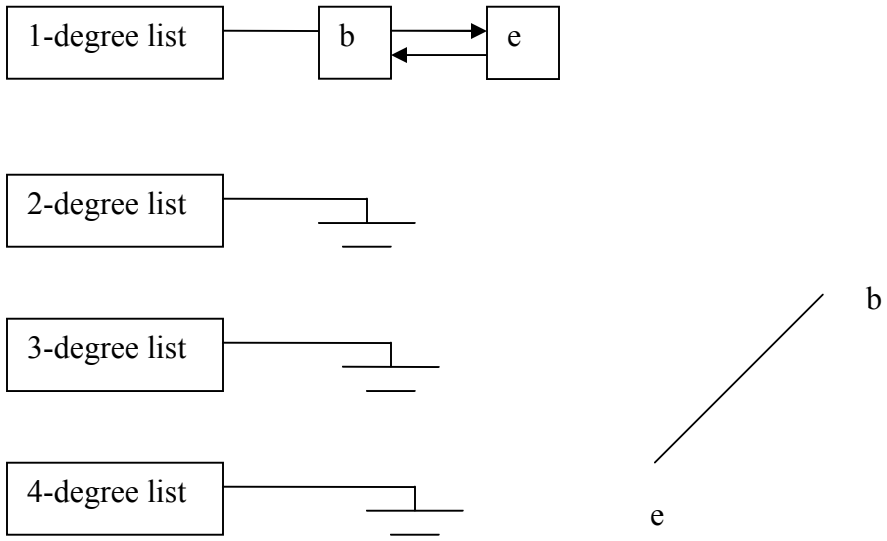

Figure 2.9. Degree List after vertex deletion. Chapter 2 figure 9 (of chapter 2)

Now $t = 1$. Extract any one of b and e and enlist in the ordered list. Let us take b. So, the minimum order list now contains a, d, c, f and b. And at last we are remaining with e.

Hence, the final minimum degree ordering looks like a, d, c, f, b, and e.

CHAPTER 3

Maximum clique Algorithm

3.1 Algorithm

Now given a graph G, we determine algorithmically what tail of a minimum degree ordering gives a maximum clique of G.

Here is the algorithm.

Given a graph G let us take $(v_1, v_2, v_3 .....v_{k-1}, v_k, v_{k+1}, v_{k+2} .....v_{n-1}, v_n)$ to be a minimum degree ordering (m.d.o.) of the vertices of G. Then

1. At first we list down the degree of each vertex in the remaining graph of the minimum degree ordering .When we delete a vertex from the graph G for determining a m.d.o ,we have to remember from which degree list we deleted the vertex.

2. We have to see which vertex has the largest degree from the right in the m.d.o.

3. From **Lemma 3** we have the last $\varphi(G)$ vertices in the minimum degree ordering form a largest clique. Also from **Lemma 2** we have all vertices in the m.d.o have degrees less than $\varphi(G)$. Hence there is no vertex in the m.d.o which has degree greater than $\varphi(G)$. Also from **Lemma 1** we have that there is at least one vertex in the m.d.o which has degree $\varphi(G)$-1.

4. Hence from 3 it follows that the vertex from the right in the m.d.o which has the maximum degree has degree $\varphi(G)$-1 in the remaining graph. From **Corollary1** if v_k to be the rightmost vertex which has maximum degree then $(v_k, v_{k+1}, v_{k+2} .....v_{n-1}, v_n)$ forms a largest clique. Let d be the maximum degree. Then we have $\varphi(G)$-1=d.

5. Hence $\varphi(G) = d + 1$ is the size of a maximum clique which starts from the vertex v_k and ends at the vertex v_n. Hence, $(v_k, v_{k+1}, v_{k+2} \ldots\ldots\ldots v_{n-1}, v_n)$ is a maximum clique having size d + 1 in the graph G.

Let us explain this with examples below.

3.2 Examples

Example 1:

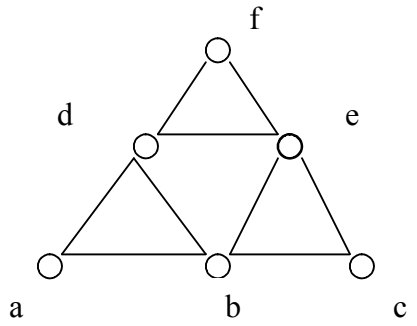

Figure 3.1. Example 1 Graph. Chapter 3 figure 1 (of chapter 3)

For the above graph we enlist the vertices and degrees of the vertices in the remaining graph in the minimum degree ordering obtained before. We have to remember from which list we deleted each vertex.

a	*f*	*d*	*c*	*e*	*b*
2	2	2	2	1	0

From the above graph we can see c has the maximum degree from the right. Hence the size of the maximum clique of the graph is $\varphi(G) = 2+ 1=3$. Hence, we have to choose last maximum degree vertex from the left and from that vertex to the end is a maximum clique. Hence, c, e, b is a maximum clique of the above graph.

Example 2

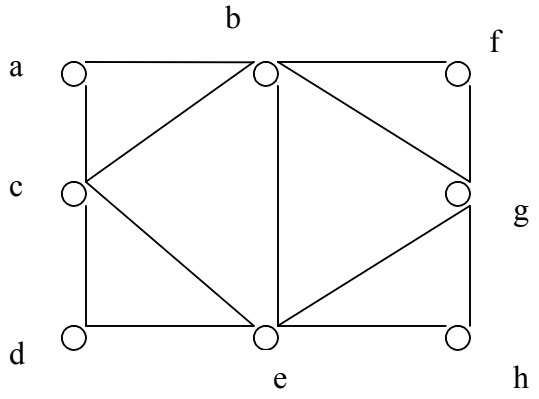

Figure 3.2. Example 2 Graph. Chapter 3 figure 2 (of chapter 3)

For the above graph we enlist the vertices and degrees of the vertices in the remaining graph in a minimum degree ordering. We have to remember from which list we deleted each vertex.

a	*f*	*h*	*g*	*b*	*c*	*d*	*e*
2	2	2	2	2	2	1	0

From the above graph we can see c has the maximum degree from the right. Hence the size of a maximum clique of the graph is $\varphi(G) = 2+ 1=3$. Hence, we have to choose last maximum degree vertex from the left and from that vertex to the end is a maximum clique. Hence, c, d, e is a maximum clique of the above graph.

Example 3

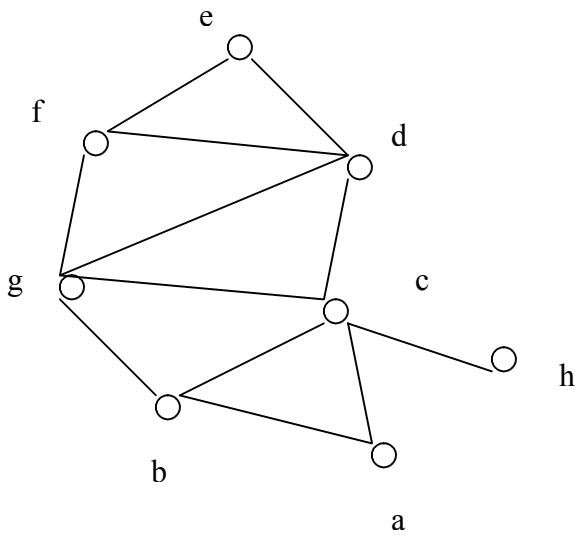

Figure 3.3. Example 3 Graph. Chapter 3 figure 3 (of chapter 3)

For the above graph we enlist the vertices and degrees of the vertices in the remaining graph in a minimum degree ordering. We have to remember from which list we deleted each vertex.

h	*a*	*b*	*c*	*g*	*d*	*f*	*e*
1	2	2	2	2	2	1	0

From the above graph we can see d has the maximum degree from the right. Hence the size of a maximum clique of the graph is $\varphi(G) = 2+ 1=3$. Hence, we have to choose last maximum degree vertex from the left and from that vertex to the end is a maximum clique. Hence, d, f, e is a maximum clique of the above graph.

CHAPTER 4

Coloring of Chordal Graphs

4.1 Definition and Algorithm

Coloring of a Graph:

Coloring a graph consists of giving a "color" (usually a number) to each vertex in such a way that adjacent vertices receive different colors.

Formally, a coloring of a graph G = (V, E) is a function $f : V \to N$ such that $f(v) \neq f(w)$ if v is adjacent to w.

Algorithm

Given a graph G and its minimum degree ordering, optimally color G with $\varphi(G)$ colors:

Input: a chordal graph G with a minimum degree

Ordering $(v_1, v_2, v_3 \ldots\ldots v_{k-1}, v_k, v_{k+1}, v_{k+2} \ldots\ldots\ldots v_{n-1}, v_n)$.

Output: a minimum coloring $c :\to \{1,\ldots\ldots,\varphi(G)\}$ and a maximum clique of size $\varphi(G)$ of G.

Algorithm:

1. Find a maximum clique in the minimum degree ordering. From Lemma 3 we know that this is the last $\varphi(G)$ vertices in the minimum degree ordering.

2. Color the last $\varphi(G)$ vertices with $\varphi(G)$ different colors. As this is a clique we must have to color each vertices of the clique with different color.

3. For each next vertex in the minimum degree ordering (from right to left) we know that degree of the next vertex is at most $\varphi(G)$-1 among right neighbors.

4. Therefore, at least one color is not used among the already colored neighbors of this current vertex.

5. We color this current vertex with that free color (color which was not present among neighbors).

6. Continuing in this way we will color all vertices using only $\varphi(G)$ colors.

7. As we can't have less than $\varphi(G)$ colors, this is a minimum coloring of the vertices of the graph G.

Now to color each vertex of the graph G we spent time deg (v). For all the vertices we spent $\sum \deg(v) = 2 \mid E \mid$. Hence we can color optimally in linear time.

4.2 Examples

Let us explain the coloring with the graph below.

As before for the graph below we enlist the vertices and degrees of the vertices in the remaining graph in a minimum degree ordering. We have to remember from which list we deleted each vertex.

1	2	3	4	5	6	7	8
2	2	2	2	3	2	1	0

From the m.d.o. we can see vertex 5 is the maximum degree (degree 3) vertex from the right. Hence the size of the largest clique is 4 and the largest clique is 5, 6, 7, 8. Hence we need 4 different colors to color the graph below. Let us take the colors as B (Blue), R (Red), Y (Yellow), G (Green). Now we have to see whether we can color the graph below with these 4 colors. We can color the largest clique 5, 6, 7, 8 with the color B, R, G and Y, respectively. Now from the algorithm above when we color the vertex 4, we have free color G and B respectively as we already colored its adjacent vertices 6 and 8 with the colors R and Y respectively. Let us color the vertex 4 with the color B. Now as

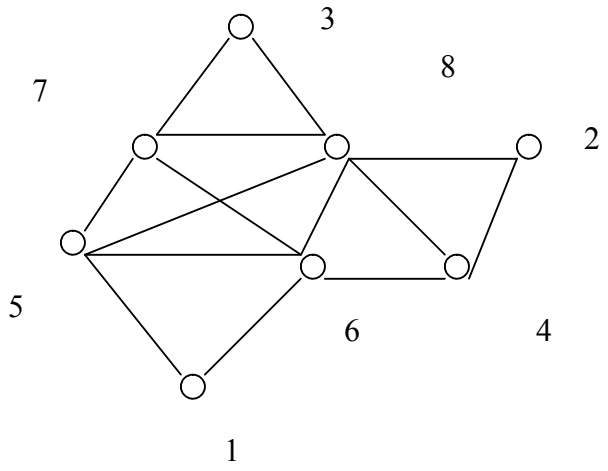

Figure 4.1. Example 1 Graph. Chapter 4 figure 1 (of chapter 4)

before we can color the vertex 3 with any one of the free color R and B. Let us color 3 with B. In this way we can color vertex 2 with the color G and vertex 1 with the color Y.

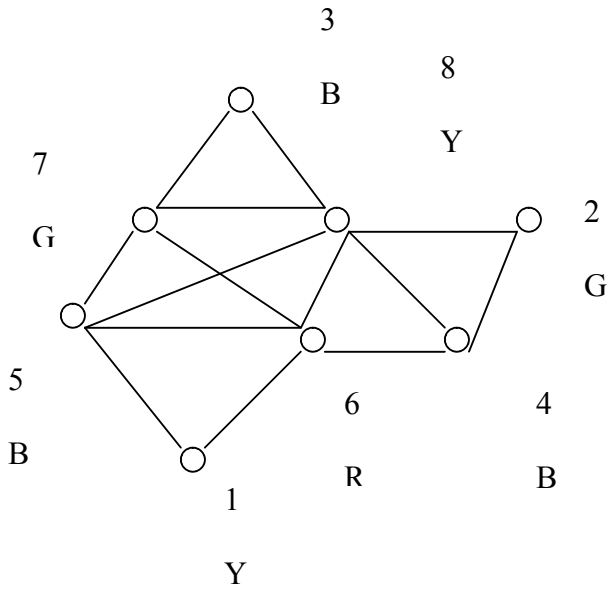

Figure 4.2. Example 2 Graph. Chapter 4 figure 2 (of chapter 4)

In this way we color the graph above with 4 colors, we cannot use less than 4 colors to color the graph as the size of the maximum clique in the graph is 4. In a clique every vertex is joined with other vertex by an edge. Hence, we have to color the vertices of the clique with distinct colors.

Let us take another example as below.

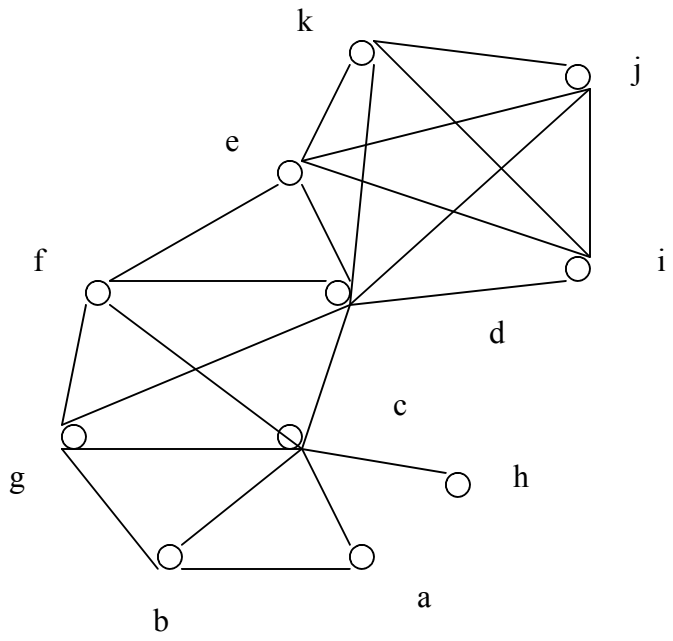

Figure 4.3. Example 3 Graph. Chapter 4 figure 3 (of chapter 4)

For this graph minimum degree ordering is.

h	*a*	*b*	*g*	*c*	*f*	*d*	*e*	*i*	*j*	*k*
1	2	2	3	2	2	4	3	2	1	0

From the m.d.o. we can see vertex d is the maximum degree (degree 4) vertex from the right. Hence the size of the largest clique is 5 and the largest clique is d, e, i, j, k. Hence we need 5 different colors to color the graph below. Let us take the colors as B(Blue),R(Red), Y(Yellow) , G (Green) and P(Pink). Now we have to see whether we can color the graph below with these 5 colors. We can color the largest clique d , e , i , j , k with the colors B , R ,G , Y and P respectively. Now from the algorithm above when

we color the vertex f, we have free colors P, Y and G as we already colored its adjacent vertices e and d with the colors R and B respectively. Let us color the vertex f with the color Y. Now as before we can color the vertex g with anyone of the free colors R, P or G. Let us color g with G. In this way we color b, c, a and h with the colors P, R, G and Y respectively. Therefore, we can color the whole graph with these 5 colors.

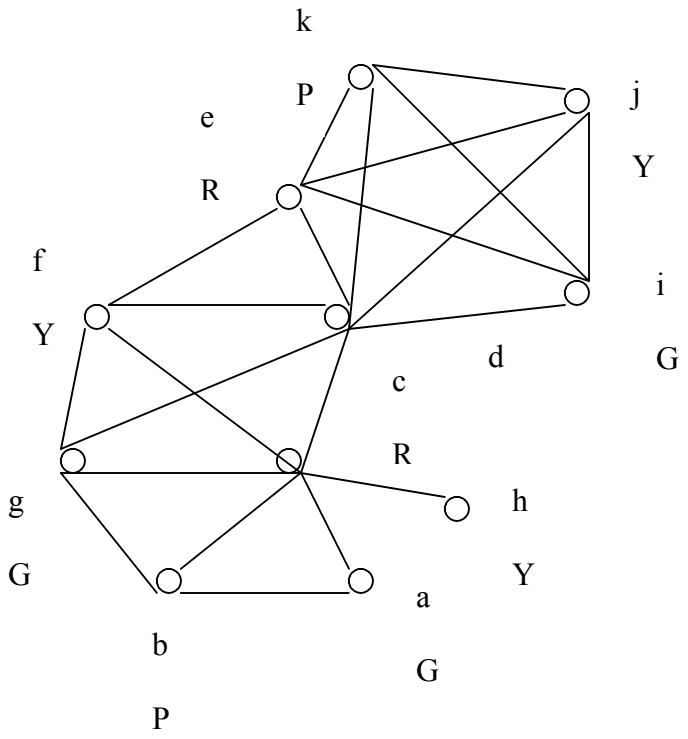

Figure 4.4. Example 4 Graph. Chapter 4 figure 4 (of chapter 4)

CHAPTER 5

Conclusion

Finding a maximum clique and a minimum coloring of a chordal graph has applications in Mathematical Science and Biology. Finding a maximum clique is an important topic in molecular cell research in Computational Biology. It has applications in Cellular Neural Networks. It is used in finding the maximum size set of intervals so that no two intervals intersect and also minimum coloring so that no two intersecting intervals have the same color. Algorithms for the maximum clique and minimum coloring problems in chordal graphs were previously given by Fanica Gavril, but in this paper we give an easier to implement and a linear time algorithm for the same problems.

BIBLIOGRAPHY

[1] Rose, D.; Lueker, George; Tarjan, Robert E. (1976). "Algorithmic aspects of vertex elimination on graphs". SIAM J. Computing 5: 266–283

[2] Gavril, Fănică (1972). “Algorithms for minimum coloring, maximum clique, minimum covering by cliques and maximum independent set of a chordal graph”, SIAM J. Computing 1, p.180-187.

[3] Bender, E. A.; Richmond, L. B.; Wormald, N. C. (1985). "Almost all chordal graphs split". *J. Austral. Math. Soc., Series A* **38** (2).

[4] Martin Charles Golumbic, "Algorithmic Graph Theory and Perfect Graphs" Academic Press, New York, 1980. Second edition, Annals of Discrete Mathematics 57, Elsevier, 2004.

[5] Lekkerkerker, C.G. and J.C. Bolad.1962. “Representation of a finite graph by a set of intervals on the real line”. Fund. Math. 51, 45-64.

[6] Fulkerson, D. R.; Gross, O.A. (1965). “Incidence Matrices and interval graphs”.

[7] G.A. Dirac. On rigid circuit graphs. Abh. Math. Sem. Univ. Hamburg 25 (1-2)

(1961), 71-72