7-7-2020

Generating Chordal Graphs with few Fill-in Edges: An Experimental Study

Aayushi Srivastava
University of Windsor

Generating Chordal Graphs with few Fill-in Edges: An Experimental Study

by

Aayushi Srivastava

A Thesis

Submitted to the Faculty of Graduate Studies

through the School of Computer Science

in Partial Fulfillment of the Requirements for

the Degree of Master of Science at the

University of Windsor

Windsor, Ontario, Canada

2020

Generating Chordal Graphs with few Fill-in Edges: An Experimental Study

by

Aayushi Srivastava

APPROVED BY:

Y. Aneja
Odette School of Business

D. Wu
School of Computer Science

A. Mukhopadhyay, Advisor
School of Computer Science

May 20, 2020

DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

Graph Generation aids in analysis of graphs and their properties while insinuating conjectures via counterexamples and even generating test instances for other algorithms. In certain cases, a list of graphs will deliver numerical information for enumerative problems devoid of theoretical solutions, or even supply a source from which specimen graphs may be adopted.

Any given graph can be embedded in a chordal graph by adding edges, and the resulting chordal graph is called a triangulation of the input graph i.e., contains no induced chordless cycle on four or more vertices. Tringulation is classified into Minimal and Minimum, where both the approach seems to minimize the number of edges added. A comparison has been drawn amongst LB-Triangulation, Lex-M and Minimum Degree Vertex (MDV) approaches to achieve triangulation with as few edges as possible along with respective run times. While LB-Triang and Lex-M algorithms provide minimal triangulation, MDV is an approximation for minimum triangulation.

Determining the minimum number of edges that must be added to a bipartite graph to make it a chain graph is NP-complete. We exploit this reduction to propose the a heuristic for obtaining a chain graph from a bipartite graph via chordal graph using MDV. Dirac's method of generating chordal graphs by union is modified with the help of MDV. Nearly Chordal Graphs are generated using a novel heuristic from complete graphs. Recognition algorithm for nearly chordal graphs is introduced and the relationship between weakly chordal, nearly chordal and chordal graphs is established.

DEDICATION

To my benevolent parents Mr. Amar Prakash Srivastava and Mrs. Sarla Srivastava, my mighty bolsters cum elder siblings Vaibhav Srivastava and Pallavi Srivastava, and my entire family whose belief in me served as a guiding light in this journey.

ACKNOWLEDGMENTS

This thesis stands a chance to thank many people for their timely help, support and inspiration. Firstly, I would like to express my sincere appreciation and gratitude towards Dr. Asish Mukopadhyay, my mentor and supervisor. His guidance, invaluable feedbacks and inspiring suggestions, have been the mightiest pillar in the development of the thesis and research.

I offer my sincere appreciation to the committee members, Dr.Dan Wu and Dr. Yash Aneja for their valuable comments and suggestions in writing this thesis. In addition, the entire Computer Science Faculty Members, Graduate Secretary and Technical Support Staff deserves special mention for all the support they provided throughout my graduation.

I am grateful to all my friends and research partners Zamilur, Sudiksha, Lokesh, Saurav, Anjali and Madhuri for their presence and support throughout. Finally, my deepest gratitude goes to my adorable family for their love, encouragement, belief and support. If not for them, I could have never imagined to come so far and strive harder. My mother always says, "Well begun is half done", and that is one quote which taught me to respect every opportunity. My brother Vaibhav has been my role model and always motivates me to aspire high. Thank you all for being there for me and let me live a dreamy yet responsible life.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

Graph theory is the study of graphs, which are mathematical structures used to model pairwise relation between objects based on the given distances between the pair of points. Graph generation techniques are widely studied and aims to give general ideas about graphs and their properties. [22]A list of graphs will provide numerical details for enumerative problems in the absence of theoretical solutions and gives a source from which specimen graphs can be taken in real-life problems. A graph is chordal if it contains no induced chordless cycle of four or more vertices.

1.1 The Triangulation problem

Any given graph can be embedded in a chordal graph by addition of edges, and the resulting chordal graph is a triangulation of the input graph. The triangulation is categorized into Minimal triangulation and Minimal triangulation.In Minimal triangulation, the inclusion-minimal set of edges are added whereas in minimum triangulation, fewest number of edges are added. The problem lies in finding such minimum number of edges while making the graph chordal. The graph $\{G = (V, E)\}$ is an arbitrary graph and the edges in red are added to make it chordal. The figure on the right represents Minimal triangulation and adds two edges which are minimal to that set of edges to make it chordal. While the graph on the left depicts minimum triangulation and is made chordal just by addition of one edge. The minimum triangulation is the minimum of all the minimal triangulations.

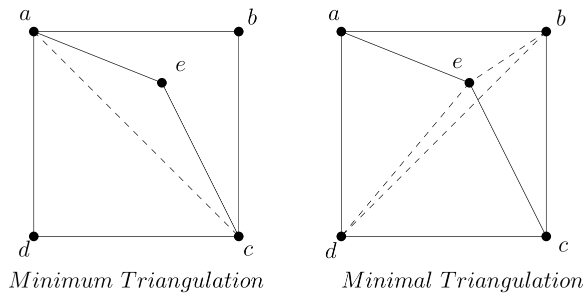

Figure 1.1: Minimal and Minimum Triangulation

1.2 Problem Statement

Triangulation problem is narrowed down towards achieving minimum triangulation which is NP-Complete [28]. The arbitrary input graph is transformed into chordal graph by addition of as few edges as possible. A graph is chordal if every cycle of length greater than three has a chord. A chord is an edge joining two non-consecutive vertices on a cycle. Several important and widely studied problems on graphs are related to computing an embedding of an arbitrary graph into a chordal graph with various properties.Any given graph can be transformed into chordal graph by addition of edges to the input graph known as triangulation. Various triangulations exists for a given graph. There are two types of triangulations discussed namely Minimal triangulation and Minimum triangulation. It is focused on generating chordal graphs with minimum number of insertions, generating nearly chordal graphs and then transforming it into chordal graphs. A comparative study is done between the algorithm for generating minimal triangulation namely LB-Triangulation and the heuristic for minimum triangulation namely Minimum Degree vertex. We modified Dirac's method for generating chordal graphs by adding fewest possible edges. The minimum fill-in problem is shown to be NP-Complete by reduction from the problem of finding a minimum number of edges to be added to a bipartite graph to turn it into a chain graph via chordal graph using Minimum Degree Vertex heuristic. We applied Minimum Degree Vertex heuristic to bipartite graphs and compared the number of edges added using it and the one via chain graph-chordal method.We also generated nearly chordal graph using a novel heuristic from complete graphs.

1.3 Motivation

The motivation of the problem comes from the fact that any arbitrary graph can be converted into chordal graph by addition of edges. The addition can be done by choosing a vertex v_1 in any order $\{v_1, v_2, ..., v_n\}$ of the vertex set of graph and then making neighbors of v_1 a clique, removing v_1 and then continue with the next vertex in that order.

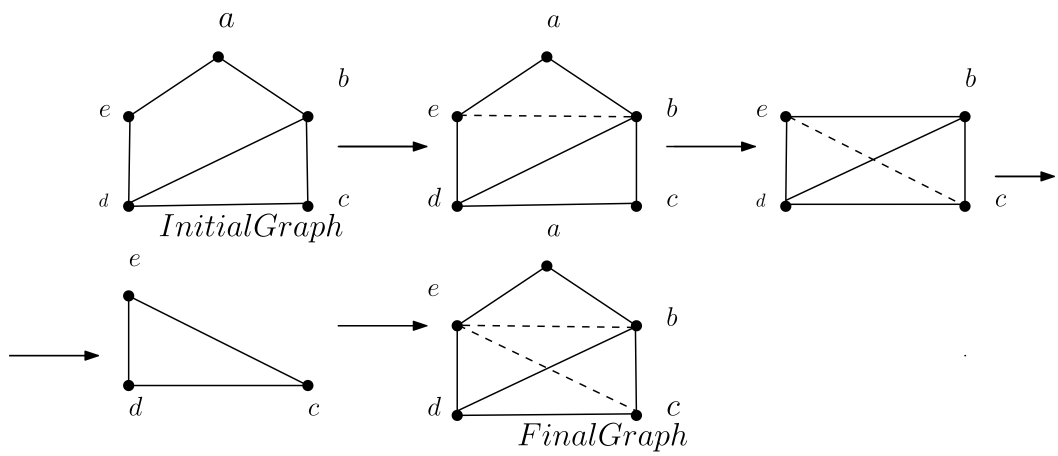

Figure 1.2: Motivation behind Triangulation Problem

The figure above explains adding the edges to an arbitrary graph by making the neighbours of vertices a simplicial, and then removing the vertex. As per the figure above, this technique could transform any arbitrary graph to chordal. Our focus narrowed down to the scenarios where we could put a check on the number of edges added in the process. The need of adding fewer edges could be fulfilled either by minimal or minimum triangulation. This encouraged us to study and compare various triangulation algorithms.

1.4 Thesis Organization

The list below presents the organization of the chapters which makes up this thesis. Also given is a brief description of the topics each chapter deals with.

- Chapter 1 gives a clear background knowledge on the Chordal Graphs, Minimal triangulation, Minimum triangulation.
- Chapter 2 A comparative study is done amongst three chordal graph generation techniques namely, LB - Triangulation, Lex - M and Minimum Degree Vertex.
- Chapter 3 Modified Dirac's method of Chordal Graph Generation by union of chordal graphs using Minimum Degree Vertex heuristic.
- Chapter 4 Reduction to propose the heuristic for obtaining a chain graph from a bipartite graph via chordal graph using MDV. Generating Chordal graphs from Bipartite Graphs using reduction method and MDV, then comparing the results of both.
- Chapter 5 Generation of Nearly Chordal Graphs, Recognition algorithm for chordal graphs and explored the relationship amongst chordal, weakly chordal and nearly chordal graphs.
- Chapter 6 Concludes the work done in this thesis and suggests some possible future research directions.
- Bibliography declares a detailed list of references from which facts and numbers have been used as a guide for this thesis.

1.5 Preliminaries

The following section gives a background details of graph terminologies,chordal graphs and its properties and followed by different algorithmic approaches to solve the triangulation problem.

1.5.1 Graph Terminologies

A graph is an ordered pair $G = (V, E)$ comprising a set of vertices or nodes and a collection of of pairs of vertices from V called edges of the graph.

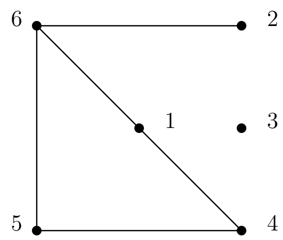

Figure 1.3: Graph[4]

$V = \{1, 2, 3, 4, 5, 6\}$

$E = (1, 4), (1, 6), (2, 6), (4, 5), (5, 6)$ [4] We let $G = (V, E)$ denote an undirected graph with vertex set V and edge set E. The number of vertices is denoted by $n = |V|$ and the number of edges by $e = |E|$. For any vertex set $S \subseteq V$, consider the edge set $E(S) \subseteq E$ given by

$E(S) := \{(u, v) \in E | u, v \in E\}$

We let $G(S)$ denote the subgraph of G induced by S, namely the subgraph of $(S, E(S))$. The graph obtained by removing a set of vertices $S \subseteq V$ from the graph is denoted by G/S.

$G/S = G(V - S)$

Two vertices $u, v \in V$ are said to be adjacent if $(u, v) \in E$. The set of vertices adjacent to v in G is denoted by $adj_G(v)$. Similarly the set of vertices adjacent to $S \subseteq V$ in G is given by:

$adj_G(S) := \{v \in V | v \notin S\text{and } (u, v) \in E \text{ for some vertex } u \in S\}$

1.5.2 What are chordal graphs?

An undirected graph $G = (V, E)$ is chordal(triangulated or rigid circuit) if every cycle of length greater than three has a chord. A chord is an edge connecting two non-consecutive vertices of the cycle. cycle.[4]

- Consider simple and connected input graph, $G = (V, E)$, with $|V| = n$ and

$|E| = m$. For a set $A \subseteq V$, $G(A)$ denotes the subgraph of G induced by vertices in A. Vertex set A is called a clique if $G(A)$ is complete.

- The process of adding edges to G between the vertices of A so that A becomes a clique in the resulting graph is called saturating A. The neighborhood of vertex v in G is $N_G(v) = \{u|uv \in E\}$, and the closed neighborhood of v is $N_G[v] = N_G(v) \cup \{v\}$.

- A vertex v is called simplicial in G if $N_G(v)$ is a clique. A vertex v is called universal in G if $N_G(v) = V \setminus \{v\}$.

- A vertex set $S \subset V$ is a separator if $G(V \setminus s)$ is disconnected. Given two vertices u and v, S is a $u - v$ separator if u and v belong to different connected components of $G(V \setminus S), S$ is then said to separate u and v. A $u - v$ separator S is minimal if no proper subset of S separates u and v.

- An elimination ordering α of G is minimal if there exists no ordering β such that G^+_β is a proper subgraph of G^+_α. [15]

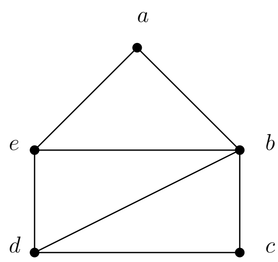

Figure 1.4: Chordal Graph

1.5.3 Properties of Chordal Graphs

The properties of Chordal Graphs which were relevant in the thesis are discussed as below:

1.5.3.1 Minimal Vertex Separators

A graph G is chordal if and only if every minimal vertex separator of G is complete in G.[8]

A subset $S \sqsubset V$ is called a $u - v$ separator of G if in $G - S$, the vertices u and v are in two different connected components. A $u - v$ separator is minimal if no proper subset of it is a $u - v$ separator.

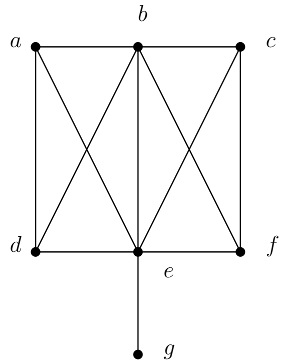

Figure 1.5: Minimal $d - c$ separator $/b, e/$ is not a minimal separator of G

In the figure 2.3, the set $S = \{b, e\}$ is a minimal $d - c$ separator; nevertheless, S is not a minimal separator of G.

1.5.3.2 Perfect Elimination Ordering

Theorem 1.5.1. *An undirected graph G is chordal if and only if it has perfect elimination ordering.[9, 13]*

A perfect elimination ordering (PEO) in a graph is an ordering of the vertices of the graph such that, for every vertex v, v and the neighbors of v that occur after v in the order form a clique.

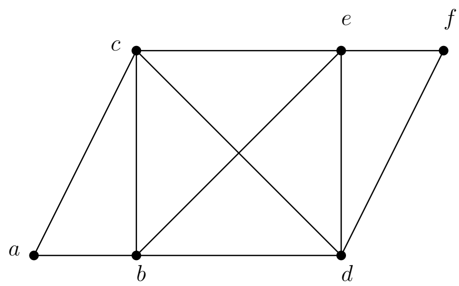

Figure 1.6: a, b, c, d, e, f is PEO

A vertex v is simplicial if $adj(v)$ induces a complete subgraph of G. The ordering α is a perfect elimination ordering (PEO) if for $1 \leq i \leq n$, the vertex v_i is simplicial in the graph G.

1.5.3.3 Maximum Cardinality Search

Theorem 1.5.2. *Every maximum cardinality search ordering of a chordal graph G is a perfect elimination ordering.[26, 25]*

PEO can be computed by using Maximum Cardinality Search (MCS). The MCS algorithm orders the vertices in reverse order beginning with an arbitrary vertex $v \in V$ for which it sets $\alpha(v) = n$. At each step the algorithm selects as the next vertex to label an unlabelled vertex adjacent to the largest number of labelled vertices, with ties broken arbitrarily. Perfect Elimination Ordering is the reverse of the ordering computed by Maximum Cardinality Search.

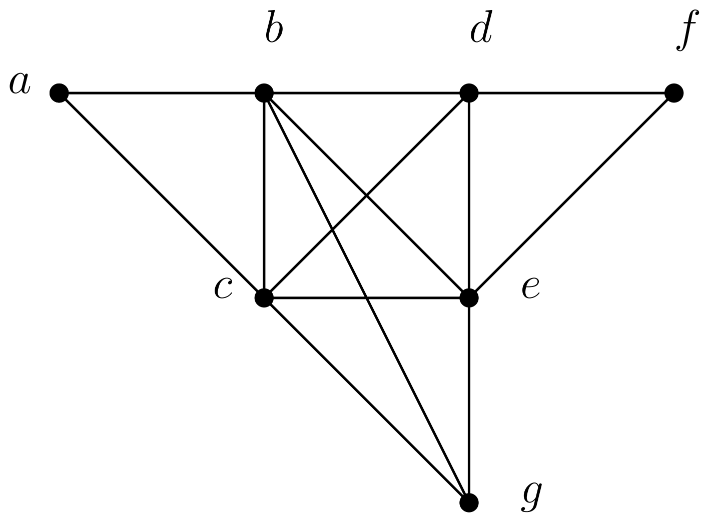

Figure 1.7: b, c, d, e, g, f, a is computed by MCS. a, f, g, e, d, c, b is PEO

$L_8 = \phi$, $L_7 = \{b\}$, $L_6 = \{b, c\}$, $L_5 = \{b, c, d\}$, $L_4 = \{b, c, d, e\}$ $L_3 = \{b, c, d, e, g\}$, $L_2 = \{b, c, d, e, g, f\}$, $L_1 = \{b, c, d, e, g, f, a\}$

Chapter 2

Minimal and Minimum Triangulations: A Comparitive Study

Tringulation(or Fill-in) is classified into Minimal and Minimum, where both the approaches aims to minimize the number of edges added. In minimal triangulation, the edges added are inclusion minimal set of edges while, minimum triangulation requires the set of edges added to be of the smallest size.A triangulation H is minimal if and only if the removal of any single fill edge from H results in a non-chordal graph.[23] The number of edges in minimal triangulation can be far from minimum and the computing of minimum triangulations is NP-Hard. Computing a triangulation with fewer edges is is relevant in solving sparse systems of linear equations[15]. Computing Minimal triangulation and minimum triangulation is studied by comparing LB-Triangulation, Lex-M and Minimum Degree Vertex techniques. While, Lex-M and LB-Triangulation provides minimal fill-in, Minimum Degree Vertex is an approximation algorithm of minimum fill-in.

2.1 Study of LB-Triangulation

Berry [1] introduced this algorithm that provides minimal triangulation in $O(nm)$ time, and that can furthermore create any minimal triangulations of an arbitrary graph in any order of vertices. LB- Triangulation is an efficient algorithm to compute minimal triangulation using an arbitrary ordering on the vertices.[1] In this algorithm, any ordering α on the vertices,produces minimal triangulation by adding only the necessary edges at each step, instead of making the current vertex simplicial.

- $G = (V, E)$ as an input graph with $|V| = n$ and $|E| = m$
- $H = (V, E + F)$ as a transitory graph which is updated at each step of algorithm with $|E + F| = m'$.

Algorithm 1: LB-Triang Algorithm

Input: An Arbitrary Graph $G = (V, E)$

Output: A minimal fill-in F of G, A minimal triangulation $H = (V, E + F)$ of G

Choose an arbitrary order α of V **for** *each vertex x in V taken in order α of V* **do**

  Compute $N[x]$ **if** $N[x] \neq V$ **then**

    Compute the set of connected components $C_G(N_H[x])$

    for *each connected component C in $C_G(N_H[x])$* **do**

      Create complete subgraph on $N_G(C)$

2.1.1 Example of LB-Triangulation

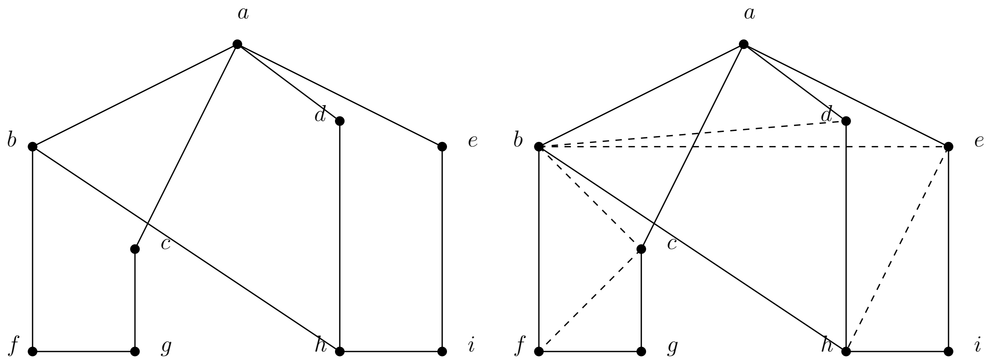

Figure 2.1: Example of LB-Triang [1]

Figure shows the minimal triangulation obtained by considering vertices in the order $(a, b, c, d, e, f, g, h, i)$ on input graph G represented by solid edges and the graph including dashed edges is the minimal triangulated graph H.

- $N_H[a] = \{a, b, c, d, e\}, C_G(N_H[a]) = \{\{f, g\}, \{h, i\}\}, N_G(\{f, g\}) = \{b, c\}$; fill-in edge bc is added; $N_G(\{h, i\}) = \{b, d, e\}$; fill-in edge $bd, beandde$ are added.

- $N_H[b] = \{a, b, c, d, e, f, h\}, C_G(N_H[b]) = \{\{g\}, \{i\}\}, N_G(\{g\}) = \{c, f\}$; fill-in edge cf is added;$N_G(\{i\}) = \{e, h\}$; fill-in edge eh is added.

2.1.2 Output of LB-Triangulation

The input graph is with black edges and has 10 nodes and 17 edges. There is an addition of four edges to make it chordal while applying the algorithm.

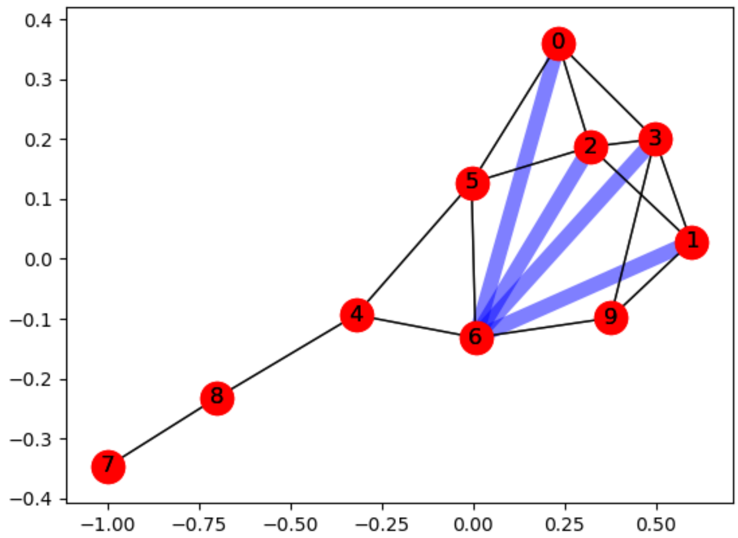

Figure 2.2: Output of LB-Triangulation Algorithm

2.1.3 Complexity of LB-Triang

The algorithm repeatedly converts any minimal separator in the neighborhood of the vertex considered into a clique, instead of making the vertex simplicial. LB-Triangulation algorithm can yield any minimal triangulation and can be implemented to run in $O(nm)$ time.

2.2 LEX-M

Lex-M produces a minimal elimination ordering(MEO) [23]. An ordering α is called minimal elimination ordering of graph G, if G^+_α is a minimal triangulation of G. Any graph G and any clique K in G, there exists a minimal elimination ordering of G where the vertices of K are numbered last, i.e. with numbers $n - |K| + 1, n - |K| + 2, ...n$. Therefore, as opposed to the first vertex of an MEO, the last vertex of an MEO can be chosen arbitrarily. [23, 15]. Thus, vertex v appends its number $\alpha(v)$ to the label of every vertex which is connected directly or a path all of whose internal vertices are unnumbered and have lexicographically smaller labels than u. Such path is called fill path. Edge uv is then an edge of the resulting minimal triangulation, and can be added to G_α^+.[15] This algorithm uses only the information from input graph G and the vertex labels during the whole process, so the added fill edges have no effect on the execution.

Algorithm 2: Lex-M [15]

Input: An arbitrary graph $G = (V, E)$

Output: A minimal elimination ordering α of G and the corresponding minimal triangulation G_α^+

$F = \phi$ **for** *all vertices v in G* **do**
    $l(v) = \phi$
for $i = n$ *down to* 1 **do**
    Choose an unnumbered vertex v of lexicographically maximum label;
    $\alpha(v) = i$, $S = \phi$
    for *all unnumbered vertices $uinV$* **do**
        if *there is an edge uv or a path $u, x_1, x_2, ..., x_k, v$ in G through unnumbered vertices such that $l(x_i)$ is lexicographically smaller than $l(u)$ for $1 \leq i \leq k$* **then**
            $S = S \cup \{u\}$
    for *all vertices $u \in S$* **do**
        Append i at the end of $l(u)$ **if** $uv \notin E$ **then**
            $F = F \cup \{uv\}$
$H = (V, E \cup F)$

2.2.1 Example of Lex-M

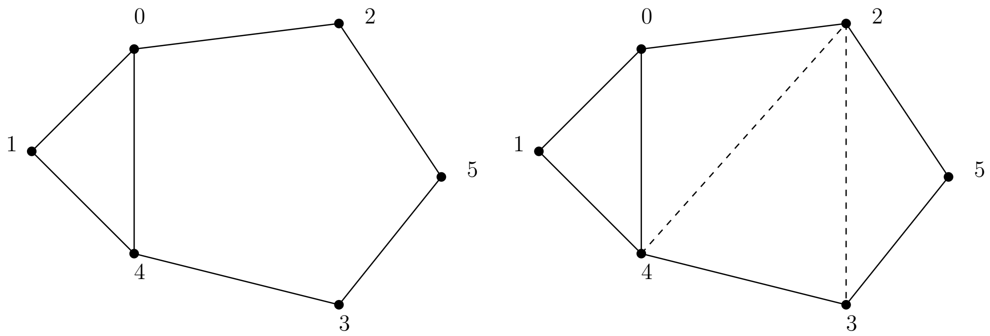

Figure 2.3: Example of Lex-M

- No. of vertices(n) = 6, No. of edges(m) = 7, $S = \phi$, $F = \phi$
- Starting with $v = 5$,and $\alpha(5) = 6$, Vertices $2, 3$ have an edge with 5, so $S = \{2, 3\}$ and labels will update $l(5) = \{\}, l(2) = \{6\}, l(3) = \{6\}$. 5 is numbered now. No new edge added.
- Now either 2 or 3 can be selected as both are of lexicographically same length, $\alpha(3) = 5$. Selected 3, already uv edge between $4, 5$ but 5 is numbered and a path $2, 0, 4, 3$ with $u = 2$, internal vertices $0, 4$ have lexicographically smaller label length than 2. $S = \{2, 4\}$. Fill -edge queue $F = \{(2, 3)\}$ is updated, labels updated:
  $l(3) = \{6\}, l(2) = \{5, 6\}, l(4) = \{5\}$. 3 is numbered now.
- Next selected $v = 2$ (maximum lexicographic label length), $\alpha = 4$.Already edge 0 and path $2, 0, 4$, so $S = \{0, 4\}$.Fill - edge queue $F = \{(2, 3), (2, 4)\}$ is updated and labels updated as $l(2) = \{5, 6\}, l(0) = \{4\}, l(4) = \{4, 5\}$.2 is numbered now.
- Next selected 4 (maximum lexicographic label length), $\alpha(4) = 3$. Already edge with 0 and 1, no path is left amongst them, so $S = \{0, 1\}$. No edge is added and labels updated as $l(4) = \{4, 5\}, l(0) = \{3, 4\}, l(1) = \{3\}$.4 is numbered.

- Next $v = 0$, $\alpha(0) = 2$ and an edge with 1, so $S = \{1\}$, no new edge added. Labels updated as: $l(0) = \{3, 4\}, l(1) = \{2, 3\}$. 0 is numbered.

- Left with $v = 1$, so given number as 1.

- Fill -edge queue is $F = \{(2, 3), (2, 4)\}$, add edges in the graph G.

- Minimal Elimination Ordering: $\{1, 0, 4, 2, 3, 5\}$

Table to summarize example results:

Vertex	Number	Label
0	2	$\{3, 4\}$
1	1	$\{2, 3\}$
2	4	$\{5, 6\}$
3	5	$\{6\}$
4	3	$\{4, 5\}$
5	6	$\{\}$

Table 2.1: Lex-M Example Summary

2.2.2 Output of Lex-M

The input graph is with black edges of 6 and 7 edges. There is addition of two edges to make it chordal while applying the algorithm.

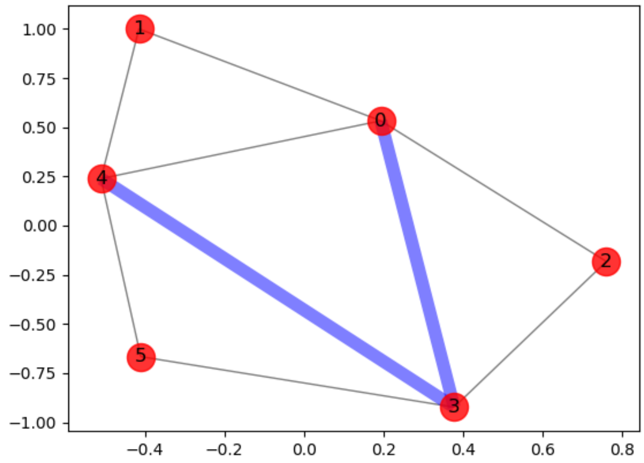

Figure 2.4: Output of Lex-M

2.2.3 Complexity of Lex-M

This algorithm constructs an ordering α for initially unordered graph $G = (V, E)$ and constructs a label $l(v)$ given by the final value of $label(v)$ for each $v \in V$. The complexity of Lex-M is $O(nm)$[1]. This time bound is not followed immediately since it has n main steps, and $O(n + m)$ time is required to follow all fill paths from the processed vertex.

2.3 Minimum Degree Vertex

Minimum Degree method chooses a vertex v of minimum degree in G^{i-1} at each step i. Minimum Degree Vertex heuristic is in the lookout of addition of as few edges as possible during triangulation. The input graph is denoted as $G = (V, E)$, with $|V| = n$ and $|E| = m$. The transitory graph obtained at the end of each step is denoted by $H = (V, E + Q)$ where Q is list of Fill-edges. We remove the vertices with degree say D, $D < 2$ (i.e. number of edges incident on it is less than 2). Then, the remaining vertices with $D \geq 2$ are arranged in the ascending order and considered in the same order. In the case where two or more vertices are of same degree, one of them is chosen first arbitrarily. We take a vertex v from the reduced vertex set and find its neighbors, make the neighbourhood a clique and then remove the vertex from graph H. The degrees of the remaining vertices in graph H are updated. We will continue till all the vertices are exhausted and in the end, we will add all the edges added in graph H to the graph G. Thus, the graph becomes chordal with as few edges as possible.

Algorithm 3: Minimum Degree Vertex

Input: An arbitrary graph H

Output: Chordal Graph C, and Fill-in queue Q of the edges added

```
Consider the vertex list Vlist
Sort Vlist in ascending order of degrees
Remove the vertices v in Vlist which has degree D(v) < 2
for vertex j in set Vlist do
    Find neighbours of j
    if No edge between neighbours then
        Add an edge between them
        Edge is inserted into the list of edges Q
    else
        No edge is added.
    Remove vertex j from the graph H and consider next vertex of lowest
      degree
    Degree is updated
```

2.3.1 Example of MDV

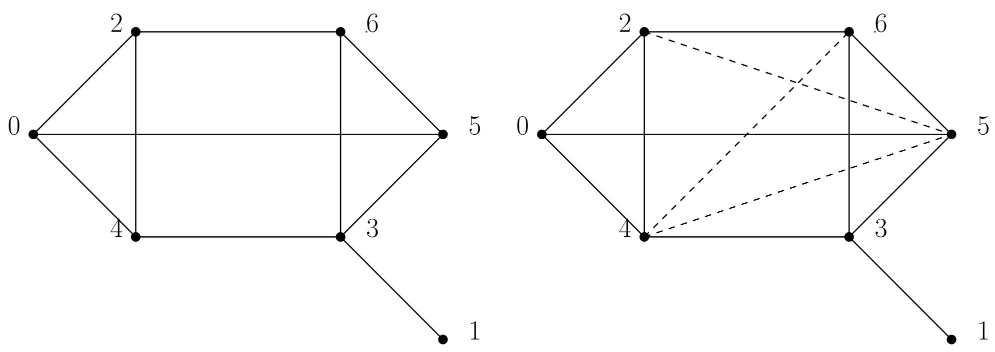

Figure 2.5: Example of MDV

- $D_H(0) = 3$, $N_H(0) = \{2, 4, 5\}$, add an edge between 2 and 5, 4 and 5, already edge 2-4 is there.
- $D_H(2) = 3$, $N_H(2) = \{4, 5, 6\}$, already edges 4-5,5-6 is present, add 4 and 6.

- $D_H(3) = 3$, $N_H(3) = \{4, 5, 6\}$, already edge 4-5,5-6,4-6 is present.

- $D_H(4) = 2$, $N_H(4) = \{5, 6\}$, edge 5-6 is already present.

2.3.2 Output of Minimum Degree Vertex

The input graph is with black edges and there is addition of two edges to make it chordal while applying the algorithm.

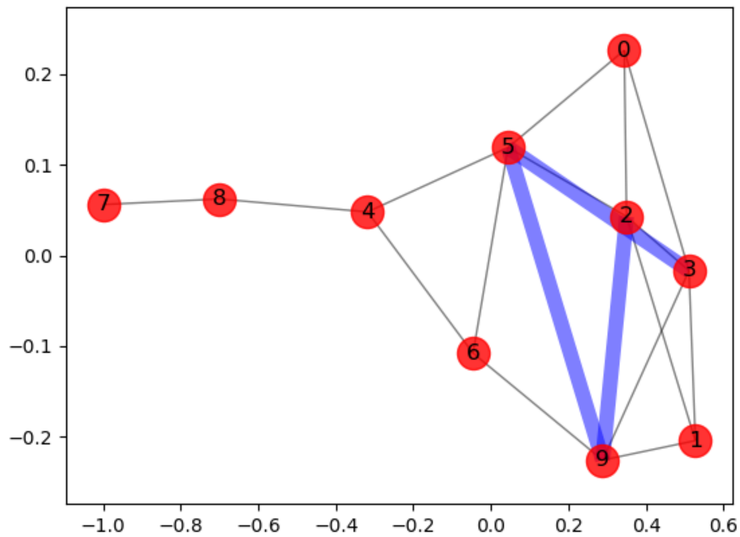

Figure 2.6: Output of Minimum Degree Vertex

2.3.3 Complexity of MDV

It is a fast algorithm and can be implemented in $O(n^2m)$ time.

2.4 Comparison between Triangulation Algorithms

We have conducted comparisons between these algorithms on the basis of the number of edges added for triangulation and the run time of algorithm.

2.4.1 Comparison between LB- triang and MDV

The comparison is done based on the number of edges added in three of them during triangulation. We have drawn comparison till graphs on 1000 vertices.

#Nodes	#Edges	#EdgesLBT	#EdgesMDV
80	100	194	93
100	150	337	191
150	200	709	275
200	250	1056	401
300	350	1576	603
400	450	1997	754
500	550	3162	1041
600	650	3626	1280
700	750	4298	1627
800	850	6177	1886
900	950	7422	2386
1000	1050	8242	2393

Table 2.2: Comparison of LB-Triang and MDV

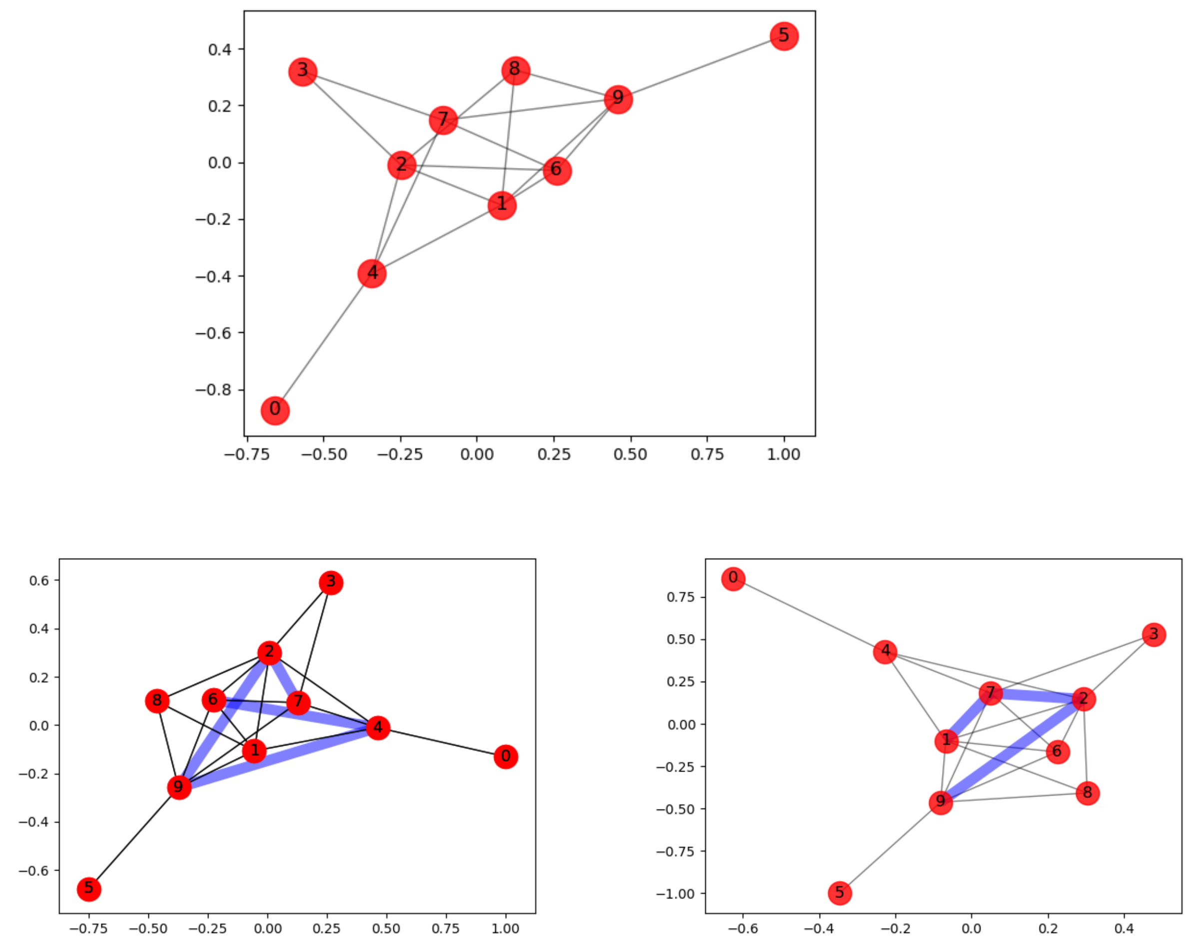

Figure 2.7: Output Comparison of LB-Triang and MDV

The input is a graph on 10 nodes and 17 edges. LB-Triangulation added 4 edges and MDV added 3 edges.

2.4.2 Comparison amongst MDV,LB-Triang and Lex-M

MDV, LB-Triang and Lex-M are compared based on number of edges added and run time. We made a table comparing edges and run time. MVD has been the quickest and added least number of edges in minimum run time. Lex-M and LB-Triang both provide minimal triangulation and added comparable edges but LB-Triang is very fast compared to Lex-M. Through the years, Lex-M has given inspiration to other minimal triangulation algorithms that have either used it or enhanced it.

$\#N$ - Number of Nodes

$\#E$ - Number of Edges

$T(s)MD$ - Run time of MDV in seconds

$EMDV$ - Edges added in Minimum Degree Vertex

$T(s)LBT$ - Run time of LB-Triang in seconds

ELM - Edges added in Lex-M

$T(s)LM$ - Run time of Lex-M in seconds

#N	#E	T(s)MD	#EMDV	T(s)LBT	#ELBT	T(s)LM	#ELM
10	15	0.005	3	0.004	5	0.020	3
15	20	0.006	3	0.007	3	0.086	4
20	25	0.008	9	0.1395	14	0.245	15
25	30	0.014	18	0.031	29	1.676	29
30	35	0.014	17	0.039	31	2.295	29
35	40	0.016	20	0.644	37	4.337	44
40	45	0.020	35	0.107	52	29.42	53
45	50	0.254	23	0.110	44	21.07	31
50	55	0.031	34	0.175	58	94.433	77
55	60	0.038	40	0.196	61	174.10	105
60	65	0.042	38	0.282	63	279.27	93
65	70	0.037	29	0.233	45	135.20	41
70	75	0.051	46	0.381	54	501.73	101
75	80	0.563	54	0.476	80	1924.64	85
80	85	0.047	37	0.521	71	1190.49	108
85	90	0.065	60	0.692	130	5961.90	115
90	95	0.054	50	0.681	111	2209.08	104
95	100	0.064	55	1.06	107	33876.62	166
100	105	0.068	61	1.316	102	37015.93	103
105	110	0.088	75	1.0730	164	12177.240	131
‘110	120	0.099	85	1.052	132	105279.911	149

Table 2.3: Comparison of MDV, LB-Triang and Lex-M

2.4.3 Output Comparison of MDV, LBT and Lex-M

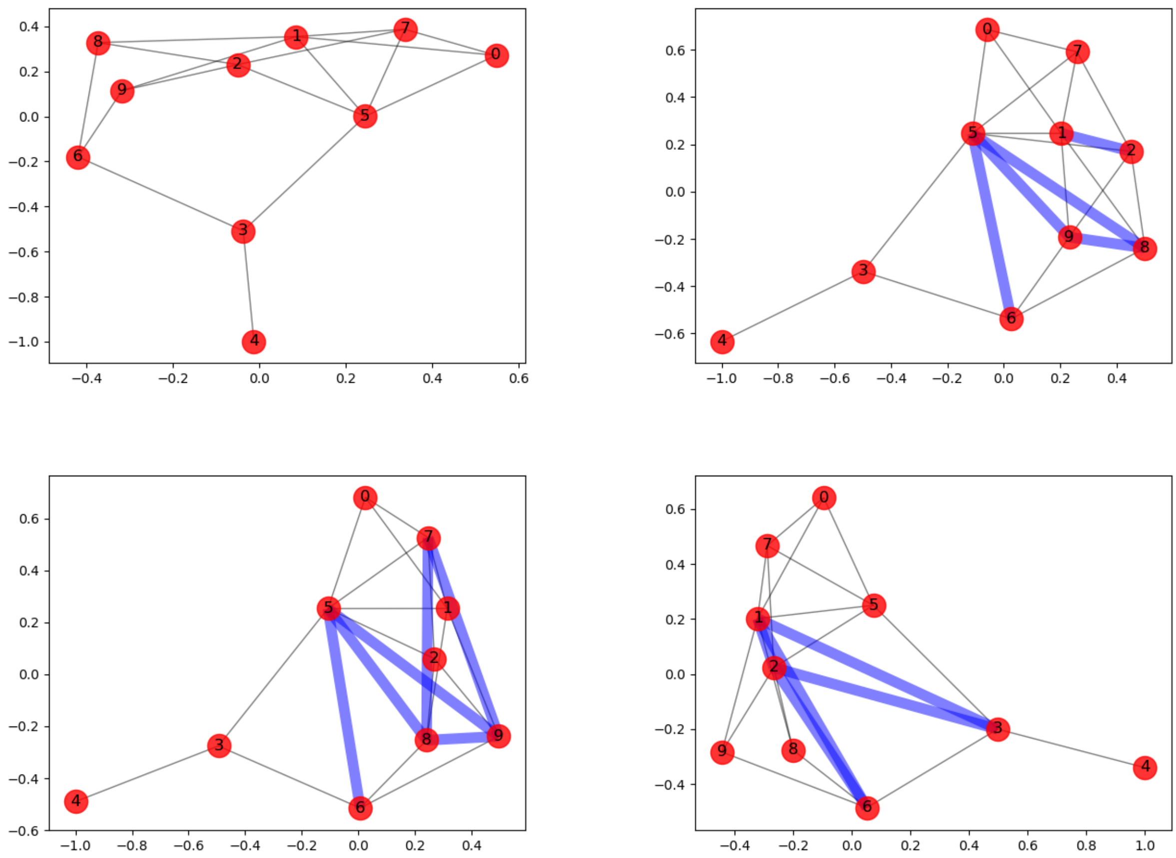

Figure 2.8: Output Comparison of MDV,LBT and Lex-M

The input graph has 10 nodes and 17 edges.

Chapter 3

Generation of Chordal Graphs by taking Union

This chapter supports the generation of Chordal Graphs by taking union of Chordal Graphs but with as few edges as possible while using Minimum Degree Vertex in it.

3.1 Definition of the Dirac's Theorem

Theorem 3.1.1. *If* Γ_1 *and* Γ_2 *are chordal graphs and* $\Gamma_1 \cap \Gamma_2$ *is a clique or empty, then* $\Gamma_1 \cup \Gamma_2$ *is a chordal graph [8].*

3.1.1 Example for the Dirac's Theorem

The below example suggests that the union of two chordal graphs whose intersection is either empty or a clique will be a chordal graph.

- First figure is a graph $G = (V, Edg\}$ with vertices as $V = \{A, B, C, D, E\}$ and edges as $Edg = \{AB, BC, CD, DE, BE, BD, AE\}$
- Graphs $BCDE$ and ABE are two chordal graphs.
- Their intersection which is EB is chordal. Hence, graph G is chordal.

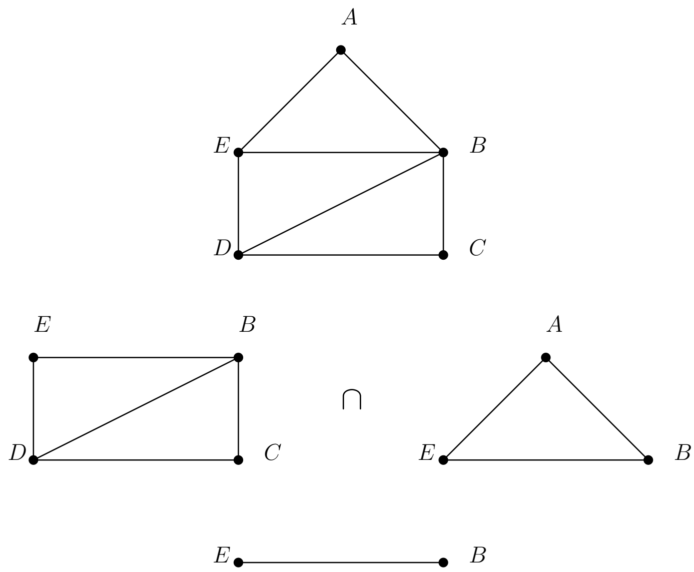

Figure 3.1: Example of Dirac's theorem

3.1.2 Counter Example of Dirac's Theorem

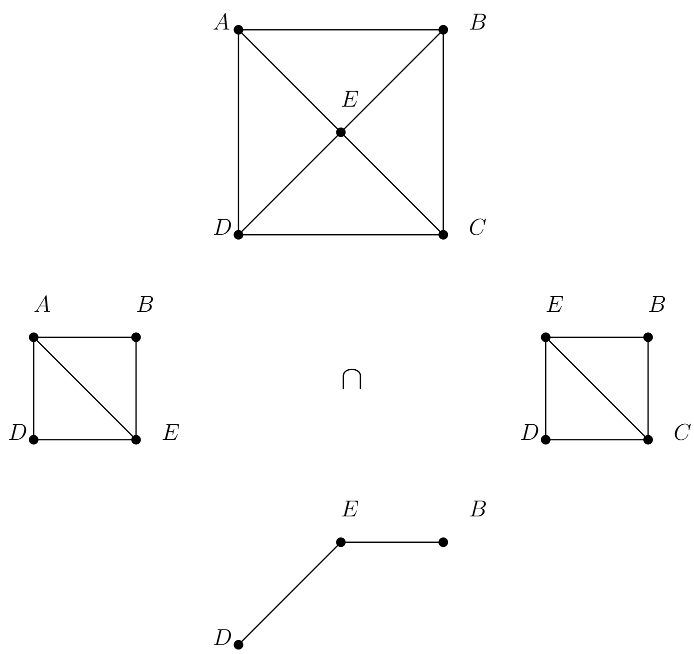

Figure 3.2: Counter Example of Dirac's theorem

We present a counter example to show that if the overlap is not a clique, the union of two chordal graphs is not chordal. The process involves three steps mentioned as below:

- First figure is a graph $G = (V, Edg\}$ with vertices as $V = \{A, B, C, D, E\}$ and edges as $Edg = \{AB, BC, CD, AD, AE, BE, DE, CE\}$
- Graphs $ABDE$ and $EBDC$ are two chordal graphs.
- Their intersection which is DEB is neither empty nor clique. Hence, graph G is not chordal.

3.2 Generation of K-chromatic Chordal Graphs

The process of generating K-Chromatic chordal graphs is as follows:

- Step 1: Generate an arbitrary graph $G = (V, E)$ with $|V| = n$ and $|E| = m$.
- Step 2: Finding mutually independent vertices M from the graph G, make them chordal say H by making such vertices simplicial and then remove from the input graph.
- Step 3: Apply Minimum Degree Vertex heuristic to the non-mutual set of vertices and make graph chordal, say C.
- Step 4: Combine graphs in Step 2 and 3 i.e., $H \cup C$ and check for chordality and name this graph as I
- Step 5: Perform minimum vertex coloring algorithm on the graph I. (Additional Feature)

3.2.1 Finding mutually independent vertices

A maximal independent set is found in graph G using networkx. An independent set is a set of nodes such that the subgraph of G induced by these nodes contains no edges. A maximal independent set is an independent set such that it is not possible to add a new node and still get an independent set. The neighbors of vertices in maximal independent set are found and made clique. After making the neighbors clique, we will remove mutually independent vertices from graph G.

Algorithm 4: Mutually Independent Vertices

Input: An arbitrary graph G with number of vertices and number of edges

Output: Graph H with edges added using mutually independent vertices

```
Using nx.maximal_independent_set find all the mutually independent
  vertices mi and put them in list Mi_List
for vertex mi in set Mi_List do
    Find neighbours of mi
    if No edge between neighbours then
        Add an edge between them
    else
        No edge is added
Remove all the mutually independent vertices from the graph
```

3.2.2 Applying Minimum Degree Vertex

After removing mutually independent vertices, perform triangulation on the remaining graph consisting of non-mutual set of vertices using Minimum Degree Vertex to make the graph chordal. The Dirac's theorem [8] still holds as the neighborhood of each vertex in the independent set being an induced subgraph of a chordal graph is also chordal.[21]

Algorithm 5: Minimum Degree Vertex for K-Chromatic

Input: A partial chordal graph H

Output: Chordal Graph C

```
Make vertex list excluding mutually independent vertices namely wmi_list
Sort the wmi_list in ascending order of degrees
Remove the vertices v which has degree D(v) < 2 for vertex j in set
  wmi_list do
    Find neighbours of j
    if No edge between neighbours then
        Add an edge between them
    else
        No edge is added
    Remove vertex j from the graph H and consider next vertex of lowest
      degree
    Degree is updated
```

3.2.3 Minimum Vertex Coloring Algorithm - Welsch Powell

The output from the combination of above two algorithms is a chordal graph. A vertex coloring is an assignment of labels or colors to each vertex of a graph such that no edge connects two identically colored vertices. A vertex coloring that minimize the number of colors needed for a given graph G is known as a minimum vertex coloring of G. In other words, no two adjacent vertices will be of the same color. The minimum number of colors itself is called the chromatic number, denoted $\chi(G)$, and a graph with chromatic number $\chi(G) = k$ is said to be a k-chromatic graph. We use Welsch-Powell algorithm for minimum vertex coloring.[11]

- Step 1. Find the degree of each vertex.

- Step 2. List the vertices in the order of descending degrees.
- Step 3. Color the first vertex with color 1.
- Step 4. Move down the list and color all the vertices not connected to the colored vertex, with the same color.
- Step 5. Repeat step 4 on all uncolored vertices with a new color, in descending order of degrees until all the vertices are colored.

Algorithm 6: Minimum Vertex Coloring

Input: A chordal graph C

Output: Chordal Graph with colored vertices

1 Find the degree $D(v)$ of all the vertices v in vertex set V

2 Sort the vertex list V in descending order of degrees

3 Remove the vertices v which has degree $D(v) < 2$ **for** *vertex j in set V* **do**

4   Color the first vertex with color 1 Check the other vertices in the list, use the same color if they are not connected

5 Check the next uncolored vertex in that order and repeat the same

---

3.2.4 Example of K-Chromatic Chordal Graph

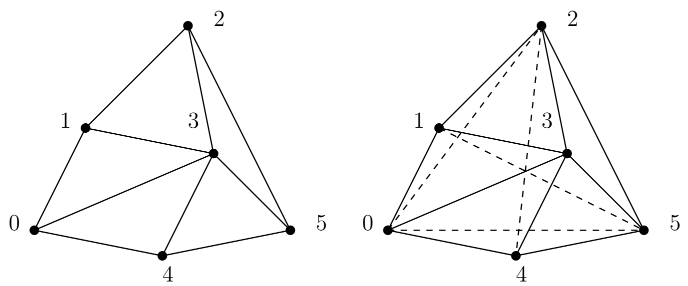

Figure 3.3: Example of K-Chromatic Chordal Graph

In the example above, Number of nodes ($n = 6$) and Number of Edges ($m = 10$).

- Mutually independent vertex is only [3], neighbors i.e. $N(3) = \{0, 1, 2, 4, 5\}$. Edges are added between $\{(0, 2), (0, 5), (1, 5), (2, 4), (2, 5)\}$ and already edges are present between $(0, 1), (0, 4), (1, 2), (1, 4), (4, 5)$. 5 edges are added. Removed 3 from graph and consider remaining graph.

- Perform Minimum Degree Vertex for remaining graph, arrange all the vertices in ascending order based on their degrees while ignoring vertices with $Degree < 2$.
- Selected 0, $N(0) = [1, 2, 4, 5]$, already edges are amongst them. Remove 0 and update degrees.
- Selected 1, $N(1) = [2, 4, 5]$, already edges are amongst them. Remove 1 and update degrees.
- Selected 2, $N(2) = [4, 5]$, already edges are amongst them. Remove 2 and update degrees.
- Selected 4, $N(4) = [5]$, already edges are amongst them. Remove 4 and update degrees.
- Graph becomes chordal now with addition of 5 edges.
- Welsch Algorithm is applied and all the six vertices gets six different color.

3.2.5 Output of K-Chromatic Chordal Graph

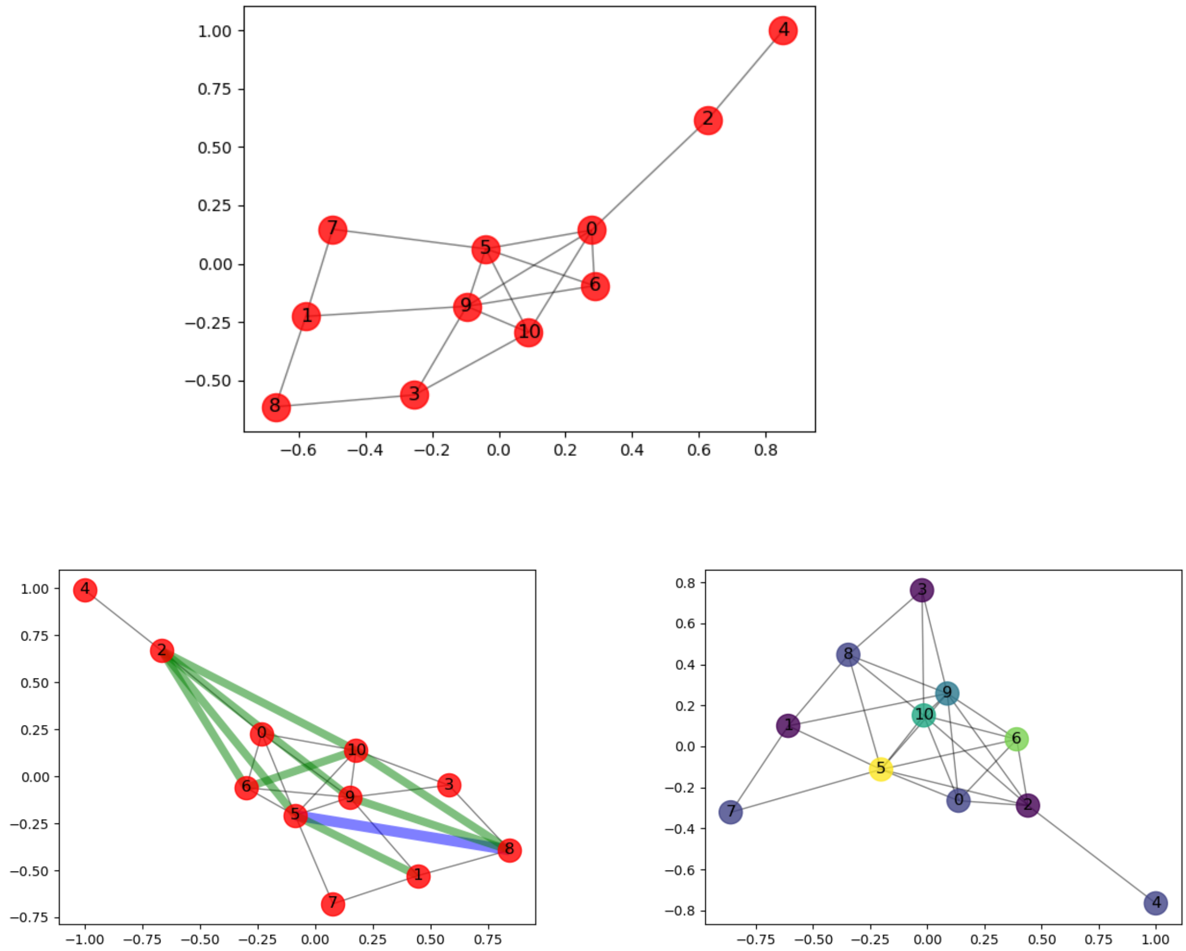

Figure 3.4: Output of K-Chromatic Chordal Graph with MDV

The green edges are added to make graph chordal via Mutually Independent Vertices Method and the blue edges using MDV and in the following figure Minimum Vertex Coloring is applied. Number of nodes $n = 11$ and Number of Edges $m = 18$ in the input graph.8 edges are added using Mutually Independent Vertices and 1 edge is added using MDV. The last graph represents an additional feature of Minimum Vertex coloring and colors 11 nodes using 6 colors.

Chapter 4

Conversion from Bipartite Graphs

4.0.1 Bipartite Graphs

A bipartite graph $G = (P, Q, E)$ is a graph whose vertices can be divided into two disjoint and independent sets P and Q such that every edge connects a vertex in P to one in Q.[28]

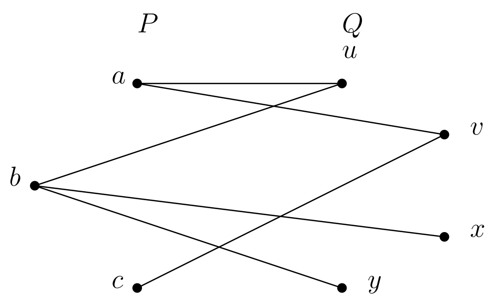

Figure 4.1: Example of Bipartite Graph

4.0.2 Chain Graphs

The bipartite graph (U, V, E) is a chain graph if the neighborhoods of the nodes in U form a chain i.e. there is a bijection $\pi : 1, 2, ..., |P|$ an ordering of P such that $\Gamma(\pi(1)) \sqsupseteq \Gamma(\pi(2)) \sqsupseteq \Gamma(\pi(3)) \sqsupseteq ... \sqsupseteq \Gamma(\pi(|P|))$.[28]

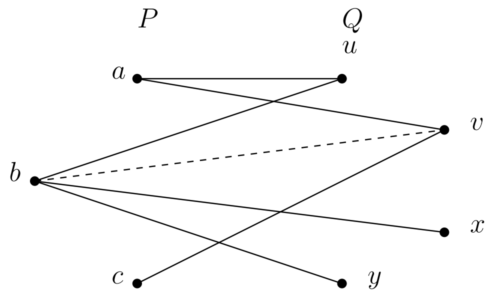

Figure 4.2: Conversion of Bipartite Graph to Chain Graph

4.1 Bipartite to Chain Graph via Chordal Graph using MDV

- Determining the minimum number of edges that must be added to a bipartite graph to make it a chain graph is NP-complete.
- By reduction from this problem it can be shown that the minimum fill-in problem for chordal graphs is NP-complete.
- We exploit this reduction to propose the following heuristic for obtaining a chain graph from a chordal graph.
- Use the MDV heuristic to add few fill-in edges to reduce an appropriate graph (C(G) in the lemma above) to a chordal graph.

Lemma 4.1.1. *Let G be a bipartite graph. $C(G)$ is chordal if and only if does not contain a pair of independent edges.[28]*

It supports that computing minimum fill-in is NP-Complete. We will convert bipartite to chain graph via chordal graph with as few edges as possible.

- Step 1: Consider a bipartite graph, $G = (P, Q, E)$.
- Step 2: Make P and Q as cliques.
- Step 3: To the graph obtained, apply Minimum degree Vertex heuristic.
- Step 4: Remove the fill-edges added while making P and Q cliques. The graph obtained is a chain graph.

4.1.1 Step 1: Consider a Bipartite Graph

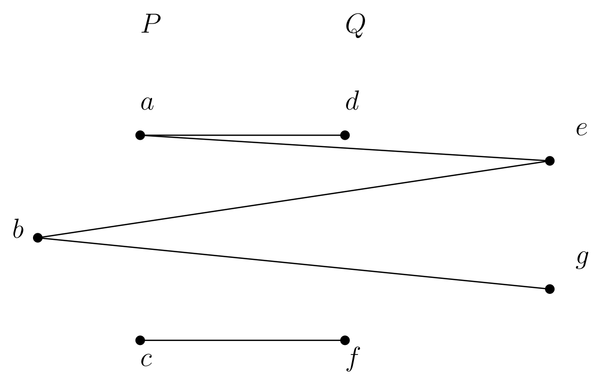

Figure 4.3: P and Q as cliques

A Bipartite graph $G = (P, Q, E)$ is taken with $P = \{a, b, c\}$ and $Q = \{d, e, g, f\}$ and $E = \{ad, ae, be, bg, cf\}$.

4.1.2 Step 2: Making P and Q as cliques

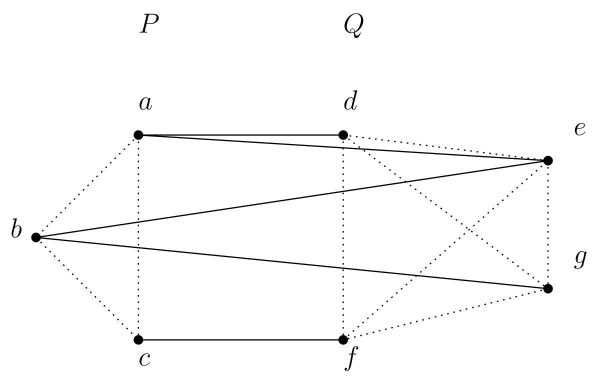

Figure 4.4: P and Q as cliques

Add edges amongst a, b, c and d, e, f, g (dotted lines) to make P and Q as cliques.

4.1.3 Step 3: Applying MDV

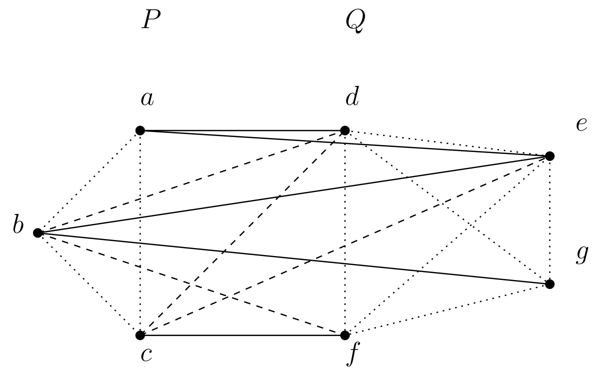

Figure 4.5: Applying MDV(dashed lines)

Tc

- $D(a) = 4$, neighbors of a are $\{b, c, d, e\}$, add edges bd, cd, ce.
- $D(g) = 4$, neighbors of g are $\{b, f, e, d\}$, add edge bf.

4.1.4 Step 4: Obtain Chain Graph

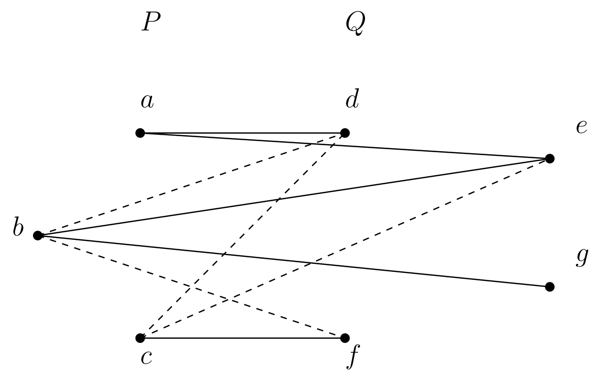

Figure 4.6: Chain Graph

We will remove edges added in P and Q which were added while making them cliques. Thus, the graph obtained is chain graph.

4.2 Conversion of Bipartite to Chordal Graph-Reduction

The problem of computing minimum triangulation is NP-Hard but it is shown to be NP-Complete for Bipartite graphs using reduction method.

Chain graphs are used for reduction. Two edges $(u, v), (x, y)$ are said to be independent in a graph G if the nodes u, v, x, y are distinct and the subgraph of G induced by them consists of exactly these two edges. Yannakakis(1981) gave following lemmas for reduction:

Lemma 4.2.1. *A bipartite graph is a chain graph if and only if it does not contain a pair of independent edges.[28]*

Lemma 4.2.2. *Let G be a bipartite graph. $C(G)$ is chordal if and only if does not contain a pair of independent edges.[28]*

Lemma 4.2.3. *It is NP-Complete to find the minimum number of edges whose addition to a bipartite graph $G = (P < Q, E)$ gives a chain graph.[28]*

4.2.1 Bipartite to Chain Graph Conversion

The bipartite graph (U, V, E) is a chain graph if the neighborhoods of the nodes in U form a chain i.e. there is a bijection $\pi : 1, 2, ..., |P|$ an ordering of P such that $\Gamma(\pi(1)) \sqsupseteq \Gamma(\pi(2)) \sqsupseteq \Gamma(\pi(3)) \sqsupseteq ... \sqsupseteq \Gamma(\pi(|P|))$.[28]

Algorithm 7: Bipartite to Chain Graph

Input: A Bipartite Graph

Output: A bipartite chain graph

for *vertex* v_1 *in set* V_1 **do**

  Find degree $D(v_1)$ of each vertex

  Rank the vertices based on their degrees i.e. the one with highest degree is given the maximum rank.Store the vertex in that order in a list named as ranklist and use indices as ranks.

for *vertex* v_2 *in set* V_2 **do**

  Find neighbour of each vertex v_2 along with its rank in the list named rank list

  Assign the rank $\sigma(v_2)$ to v_2 of its neighbor of highest rank in ranklist

  if $D(v_2) < \sigma(v_2)$ **then**

    Make $D(v_2) = \sigma(v_2)$ i.e. add an edge between v_2 with the vertex in set V_1 to make these equal

  else

    No edge is added in this case

4.2.2 Example of Bipartite to Chain Graph Conversion

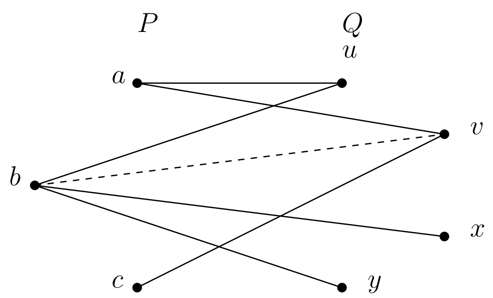

Figure 4.7: Conversion of Bipartite Graph to Chain Graph

$\Gamma(b) = \{u, v, x, y\}$, $\Gamma(a) = \{u, v\}$, $\Gamma(c) = \{v\}$

$\Gamma(b) \sqsupseteq \Gamma(a) \sqsupseteq \Gamma(c)$

Rank the vertices by decreasing vertex degree. $\sigma(b) = 1, \sigma(a) = 2, \sigma(c) = 3$

$\sigma(u) = 2$, $\sigma(v) = 3$, $\sigma(x) = 1$, $\sigma(y) = 1$

4.2.2.1 Conversion of Chain Graph to Chordal Graph

It is based on heuristic to find the node of maximum degree in both parts of the graph. Then, join such neighbors of the node in the other side of graph.

Algorithm 8: Chain to Chordal Graph

Input: A Chain Graph with number of nodes in both parts of the graph and number of edges

Output: A chordal graph with increased number of edges

```
// V1 is set of vertices in P side of graph
// V2 is set of vertices in Q side of graph
1 for vertex v1 in set V1 do
2     Find the vertex with maximum D(v1) and list its neighbors in set V2.
      Add edges amongst all the neighbors of the maximum degree v1 in
      set V2
3 for vertex v2 in set V2 do
4     Find the vertex with maximum D(v2) and list its neighbors in set V1.
      Add edges amongst all the neighbors of the maximum degree v2 in
      set V1
```

4.2.3 Example of Chain Graph to Chordal Graph Conversion

The above example shows the conversion of chain graph to chordal graph.

- In the first part of graph i.e. $\{a, b, c\}$, b has the highest degree i.e. $D(b) = 4$ and the vertices connected to b are u, v, x, y and we will add an edge between all these vertices. Edges $\{uv, ux, uy, vx, vy, xy\}$ are added.

- In the second part i.e. $\{u, v, x, y\}$, $D(v) = 3$ which is highest and its neighbors are $\{a, b, c\}$, so edges added are : $\{ab, ac, bc\}$

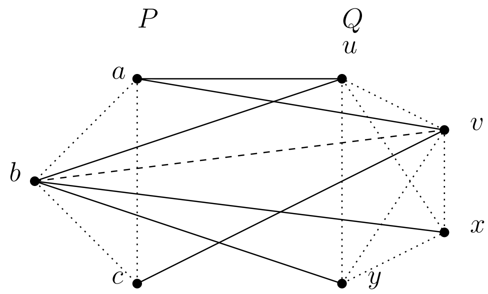

Figure 4.8: Example of Chain Graph conversion to Chordal graph

4.2.4 Output of Bipartite to Chordal graph conversion using Reduction

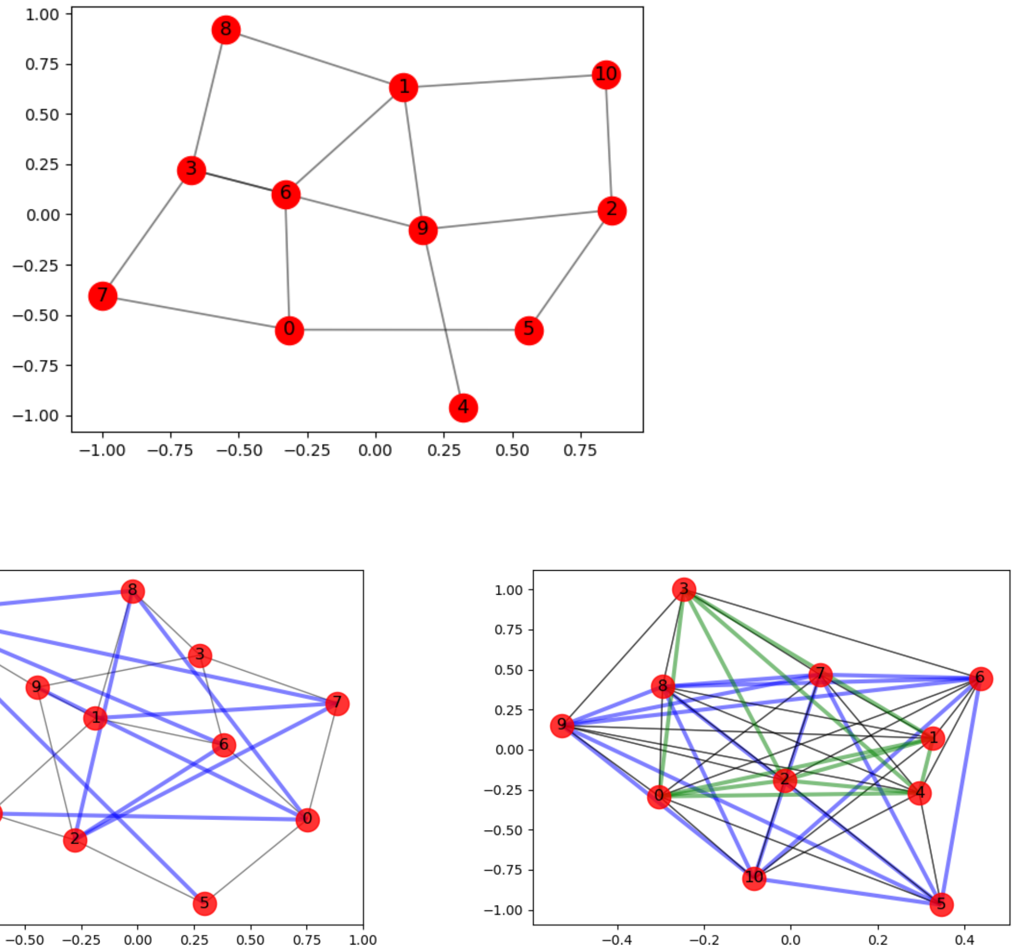

Figure 4.9: Conversion from Bipartite to Chain and then Chordal Graph on (3, 4) nodes and 8 edges

First figure is a bipartite graph with number of nodes in first part as 4 and in second part as 5 with number of edges as 12. Then, there is a chain graph with 4 blue edges required for conversion. Followed by Chordal graph with an addition 16 green edges needed for triangulation.

4.3 Conversion of Bipartite to Chordal Graph by Minimum Degree Vertex

Minimum Degree Vertex heuristic is applied on a bipartite graph. The aim is to make this bipartite graph chordal with as few edges as possible.

Algorithm 9: Minimum Degree Vertex for Bipartite Graphs

Input: A bipartite graph $G = (P, Q, E)$

Output: Chordal graph $H = (P, Q, E + F)$

Combine the vertex list of both P and Q into V.

Sort V in ascending order of degrees.

Remove the vertices v in V which has degree $D(v) < 2$

for *vertex j in V* **do**

    Find neighbours of j

    if *No edge between neighbours* **then**

        Add an edge between them

        Edge is inserted into the list of edges F

    else

        No edge is added.

    Remove vertex j from the graph G and consider next vertex of lowest degree.

    Degree is updated

4.3.1 Example of Bipartite to Chordal Graph MDV

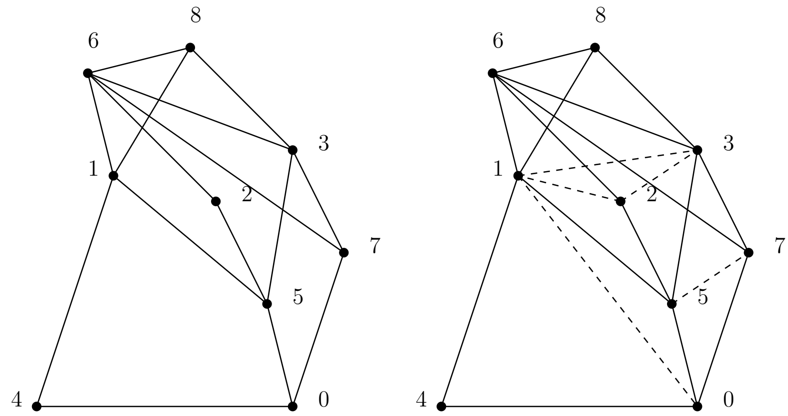

Figure 4.10: Bipartite Chordal Graph Output

- Graph $G = (P, Q, E)$, Nodes in $P = 4$, Nodes in $Q = 5$, Edges $E = 16$
- Combined vertex List i.e. $V = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$ and arrange them in ascending order of degrees and the order of vertices along with their degrees is in order of vertex followed by degree $V' = \{(0, 3),(1, 5), (2, 4), (3, 4),(5, 4), (6, 3), (7, 4), (8, 3)\}$. In case of ties in degree, one of them is chosen arbitrarily. Dictionary is updated and vertex chosen is removed every time.
- Starting with $v = 4$,Neighbors of $N(4) = [0, 1]$, will add an edge between them. Degrees are updated and 4 is removed.
- Next, 0 is selected , $N(0) = [1, 5, 7]$. $(1, 7)$ and $(1, 5)$ are already present, so edge $(5, 7)$ is added. Degrees are updated and 0 is removed.
- Next, 6 is selected, $N(6) = [1, 2, 3]$. As none of them existed beforehand, so add edges $(1, 2), (1, 3), (2, 3)$. Degree updated and removed vertex 6.
- 8 is selected, $N(8) = [1, 2, 3]$, no edges are added since they are already present and removed vertex 8.

- 1 is chosen, $N(1) = [2, 3, 5, 7]$,no edges are added since they are already present and removed vertex 1.
- 2 is selected, $N(2) = [3, 5, 7]$, no edges are added since they are already present and removed vertex 2.
- 3 is selected, $N(3) = [5, 7]$, no edges are added since they are already present and removed vertex 3.
- The graph is empty and MDV added 5 edges to the bipartite graph while making it chordal.

4.3.2 Output of Bipartite to Chordal graph MDV

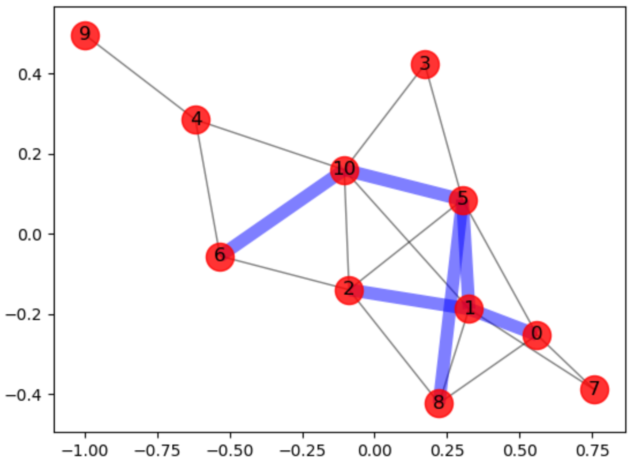

Figure 4.11: Output of Bipartite to Chordal graph using MDV

4.4 Comparison between Reduction and MDV

We will compare both the algorithms based on number of edges added and run time.

#NodesP	#NodesQ	#Edges	Time(s)MDV	#EdgesMDV	Time(s)Red	#EdgesRed
25	25	100	0.157	450	0.244	576
45	55	150	0.376	1789	3.777	2265
75	75	200	0.652	4143	20.29	5331
120	80	250	1.032	6471	59.715	8725
100	150	300	1.30	10214	150.24	13861
150	150	350	2.396	16570	325.28	20941
200	150	400	2.40	22303	716.46	29353
210	190	450	2.45	28408	1003.32	35806
225	225	500	3.96	36011	1514.72	44703
200	300	550	4.02	39747	2194.24	52285
300	250	600	5.45	57185	6864.61	70956
300	300	650	6.63	64369	5001.81	80628
350	300	700	7.448	78169	7838.11	97660

Table 4.1: Comparison of Reduction and MDV

4.4.1 Output of Comparison of Reduction and MDV

The initial inputs are: Nodes in $P = 7$, Nodes in $Q = 8$ and $Edges = 22$. Edges added in Reduction is 49 and edges added in MDV is 14.

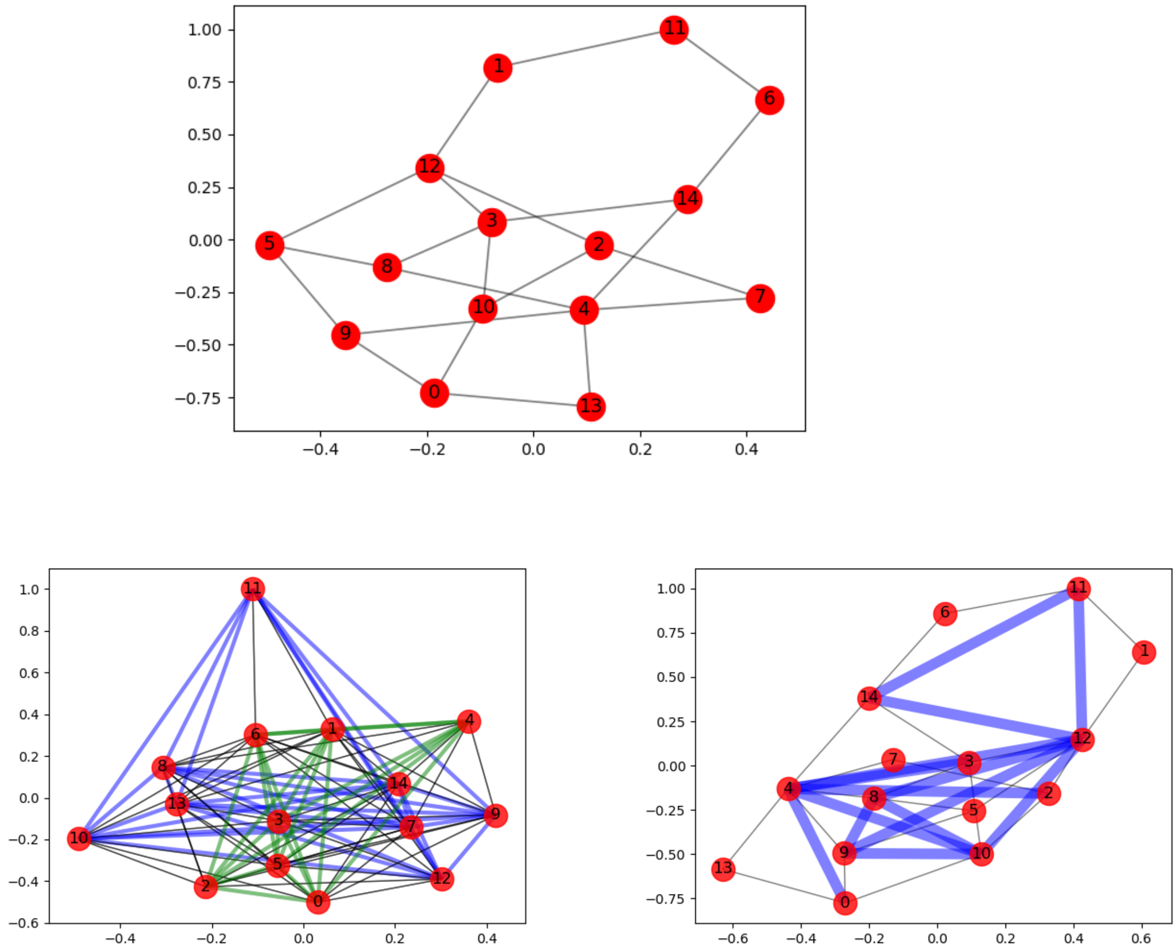

Figure 4.12: Output Comparison for Reduction and MDV

Chapter 5

Nearly Chordal Graphs

A graph is chordal if it contains no induced cycle C_k, $k \geq 4$. A graph is nearly chordal if for every vertex, the induced sub graph of its non-neighbors is chordal. [6]

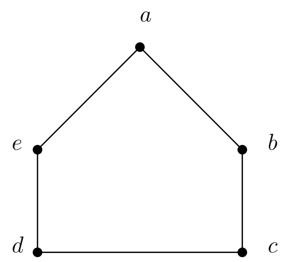

Figure 5.1: Nearly Chordal Graph

5.1 Our Approach of Generation of Nearly Chordal

We will begin with Complete graph on n nodes. A complete graph is a graph in which each pair of graph vertices are connected to each other by an edge. A complete graph of n vertices has $\binom{n}{2}$ which is $(n(n-1))/2$ edges. It is nearly chordal by default and we will make use of the basic definition of nearly chordal to convert this complete graph into nearly chordal graph. We will delete edges from complete graph while ensuring its nearly chordal property which is for every vertex, the subgraph induced by the set of its non-neighbors is chordal.

5.1.1 Methodology

Before mentioning the steps, I would liked to define the terminologies, I have used in the process. Graph $G = (V, E)$ has V as vertex set and E as edge set.

- $\{u, v\}$ is an edge with vertices as u and v to delete.
- $N(u)$ and $N(v)$ are the neighbors of vertex u and v respectively.
- $NN(u)$ and $NN(v)$ are the non-neighbors of the vertex u and v respectively.
- $G(v \cup NN(u)$ is the subgraph induced on the vertex v and non-neighbors of u. Similarly, $G(u \cup NN(v)$ is the subgraph induced on vertex u and non-neighbors of v.
- Vertex w is a set of vertices excluding the union of neighbors of u and v.
- $G(NN(w))$ is the subgraph induced on non-neighbors of the vertex w.

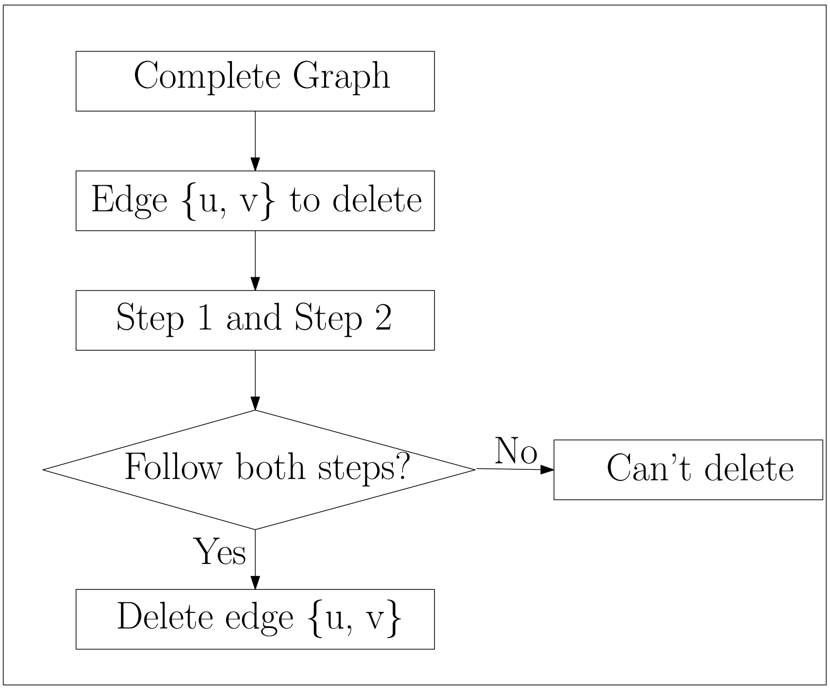

Figure 5.2: Flow Diagram for Nearly Chordal Graph Generation

Below is the criteria in two steps, we followed while conversion:

- ***Step*1 :** If we remove the edge $\{u, v\}$, where $v \in NN(u)$ and $u \in NN(v)$, check if $G(v \cup NN(u))$ and $G(u \cup NN(v))$ is chordal. If they are chordal, we may consider deleting them, followed by step 2 and if this is not true, we will add the edge $\{u, v\}$ back and no need to perform step 2.

- ***Step*2 :** Check if u and v belong to the $NN(w)$ of some w where $w \in \{V - (N(u) \cup N(v))$. If $u, v \in NN(w)$, check if removing the edge $\{u, v\}$ destroys the chordality of $G(NN(w))$.

We have applied above two steps to convert a complete graph into nearly chordal. We input the number of nodes in a complete graph, make complete graph and then select any random edge for deletion based on above two steps.

5.1.1.1 Example

In figure a to k, we demonstrate how it started with a complete graph and resulted in an empty graph. The output depends on the edges given as input and the user may exit anytime as every iteration provides nearly chordal graph.

- Figure (a) is a complete graph with 5 nodes and 10 edges.

- In Figure b, edge $\{2, 3\}$ is deleted.As we could see, $N(2) = [0, 1, 3, 4]$ and $N[3] = [0, 1, 2, 4], w = []$. Thus, $NN(2) = [3]$ and $NN(3) = [2]$ and both of them are chordal, so we can delete edge $\{2, 3\}$.

- In Figure c, edge $\{3, 4\}$ is deleted. As, $N(3) = [0, 1, 2, 3]$ and $N(4) = [0, 1, 4], w = []$. Thus, $NN(3) = [2, 4]$ and $NN(4) = [3]$, and both of them are chordal. Thus, we may delete edge $\{3, 4\}$.

- In figure d, edge $\{1, 2\}$ is deleted. As, $N(1) = [0, 2, 3, 4]$ and $N(2) = [0, 1, 4], w = []$. Thus, $NN(1) = [2]$ and $NN(2) = [1, 3]$, both of them are chordal, so we may delete this edge.

- In Figure e, edge $\{0, 2\}$ is deleted. As, $N(0) = [1, 2, 3, 4]$ and $N(2) = [0, 4], w = []$. Thus, $NN(0) = [2]$ and $NN(2) = [0, 1, 3]$, both of them are chordal, so we may delete this edge.

- In Figure f, edge $\{1,3\}$ is deleted. As, $N(1) = [0,3,4]$ and $N(3) = [0,1]$,since w is non-empty, we find $w = [2], NN(w) = [0,1,3], NN(1) = [2,3]$ and $NN(3) = [1,2,4]$, both of them are chordal, so we may delete this edge.

- In Figure g, edge $\{0,1\}$ is deleted. As, $N(0) = [1,3,4]$ and $N(1) = [0,4]$, since w is non-empty, we find $w = [2], NN(w) = [0,1,3], NN(0) = [1,2]$ and $NN(1) = [0,2,3]$, both of them are chordal, so we may delete this edge.

- In figure h, edge $\{0,4\}$ is deleted. As, $N(0) = [3,4]$ and $N(4) = [0,1,2], w = []$. Thus, $NN(0) = [1,2,4]$ and $NN(4) = [0,3]$, both of them are chordal, so we may delete this edge.

- In Figure i, edge $\{0,3\}$ is deleted. As, $N(0) = [3]$ and $N(3) = [0]$, since w is non-empty, we find $w = [1,2,4], NN(w) = [0,3], NN(0) = [1,2,4,3]$ and $NN(3) = [1,2,4,0]$, both are chordal, so we may delete this edge.

- In Figure j, edge $\{2,4\}$ is deleted. As, $N(2) = []$ and $N(4) = 1$, $NN(2) = [3,0,4,1]$ and $NN(4) = [2,3,0]$, both are chordal, so we may delete this edge.

- In Figure k, edge $\{1,4\}$ is deleted.

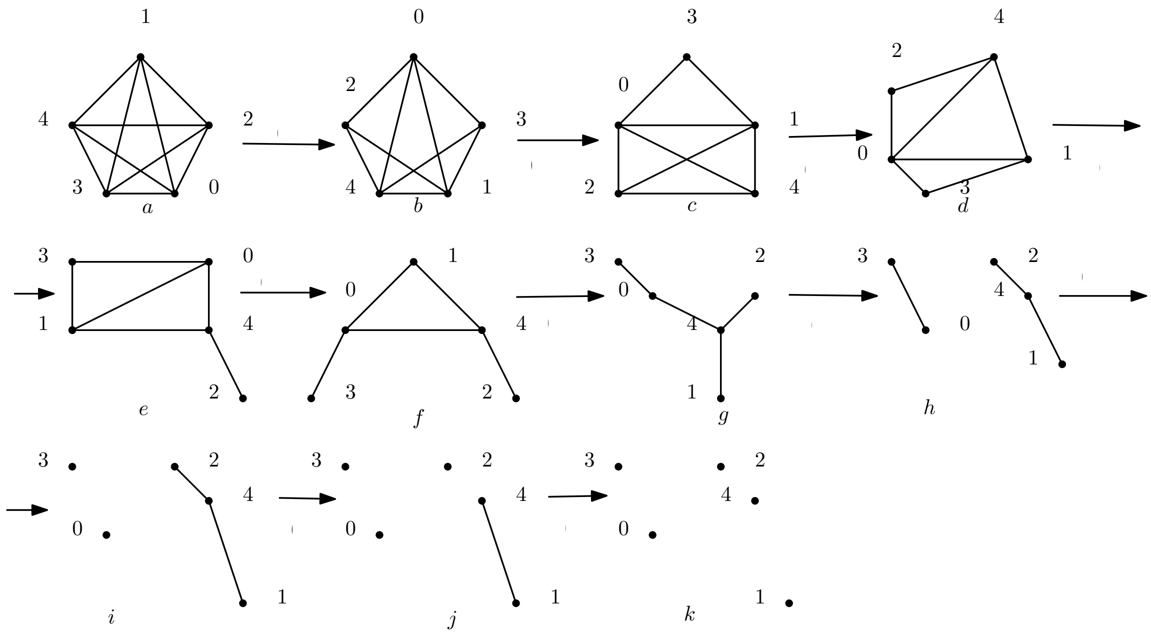

Figure 5.3: Nearly Chordal Graph Example

5.1.2 Output of Generation of Nearly Chordal Graph

In the above output, the input is a complete graph with $n = 7$ nodes. For the nearly chordal graph, it is user's discretion to select which edge(s) to delete and every deletion gives nearly chordal graph. I have randomly deleted 7 edges in the order as mentioned below: $\{(4,6),(5,1),(2,3),(0,1),(6,5),(1,3),(0,2)\}$.

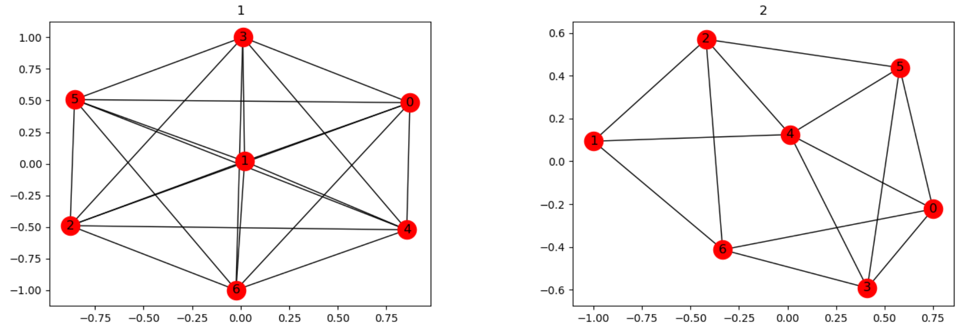

Figure 5.4: Complete Graph to Nearly Chordal Graph

5.2 Recognition Algorithm for Nearly Chordal

We have devised an algorithm to check if any arbitrary graph is nearly chordal or not. The input is number of nodes and number of edges. An arbitrary graph is made and each of its vertices are checked with the nearly chordal property.

Algorithm 10: Nearly Chordal Recognition

Input: An arbitrary graph $G = (V, E)$

Output: Check whether G is nearly chordal or not

$C = \phi$

$is\_nearly\_Chordal = False$

for *all vertex v in V* **do**

  Find non-neighbours of v, NNv Induce subgraph on non-neighbors, $S_G(NNv)$

  if *NNv is empty or $S_G(NNv)$ is chordal* **then**

    Append v to C

if $len(C) == len(V)$ **then**

  $is\_nearly\_Chordal = True$

5.3 Relationship amongst Chordal, Nearly Chordal and Weakly Chordal Graphs

A simple, undirected graph $G = (V, E)$ is said to be weakly chordal if neither G nor its complement, $\overline{G}$, has an induced chordless cycle on five or more vertices.[17].

While studying the relationships, we came to below conclusions:

- Every chordal graph is weakly chordal and nearly chordal.(Figure 5.4)

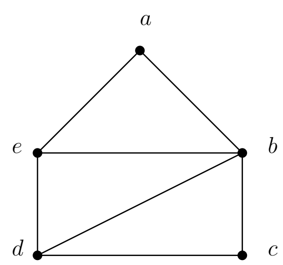

Figure 5.5: Chordal, Nearly and Weakly Chordal

- A weakly chordal graph may or may not be Chordal or Nearly Chordal. (Figure 5.5)

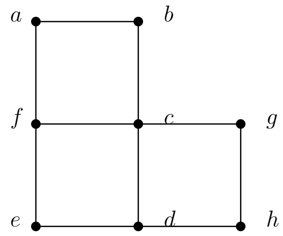

Figure 5.6: Only Weakly Chordal Graph

- A nearly chordal graph may or may not be chordal or weakly chordal.(Figure 5.6)

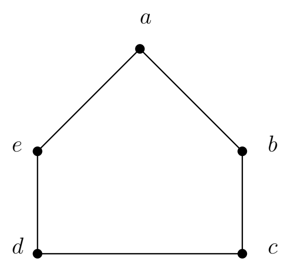

Figure 5.7: Only Nearly Chordal Graph

- A nearly chordal graph but not weakly chordal.(Figure 5.8)

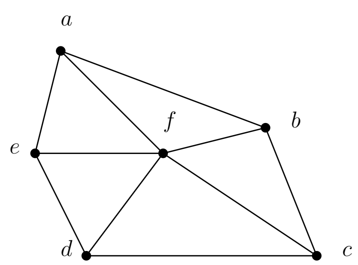

Figure 5.8: Nearly Chordal Graph but not weakly Chordal

- A nearly chordal and weakly chordal graph which is not chordal.(Figure 5.9)

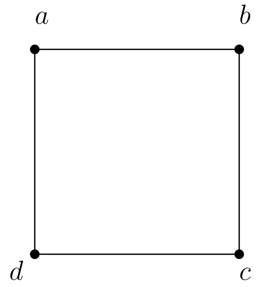

Figure 5.9: Nearly and Weakly Chordal Graph but not Chordal

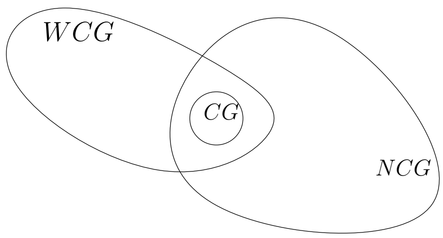

Figure 5.10: Relationship between Nearly Chordal, Weakly chordal and Chordal Graphs

In words, the relationship is depicted from above figures covering below relations:

- A Chordal Graph is both Nearly Chordal and Weakly Chordal.
- A weakly chordal graph maybe chordal or nearly chordal or both.
- Similarly, a nearly chordal graph maybe chordal or nearly chordal or both.

Chapter 6

Conclusions

This thesis contributes towards the aim of studying, implementing and comparing various triangulation algorithms. Minimal and Minimum Triangulation represents the number of fill-in edges added for to convert arbitrary graph into chordal. We have generated chordal graphs with fewest edges possible.

Chapter 2 throws ample light on the minimal triangulation algorithms including Lex-M and LB-Triangulation along with minimum triangulation heuristic of Minimum Degree Vertex Algorithm (MDV). MDV has been an efficient approximation algorithm which we have used in various other triangulation methods.The comparison amongst these three are done on the basis of the number of edges added in an arbitrary graph while triangulation. Chapter 3 focuses on generation of chordal graphs by taking union of two chordal graphs. Dirac's Method has been modified in order to achieve triangulation with fewest possible edges. The triangulation is performed on union by segregating vertices into Mutually Independent and non-mutually independent set. The vertices which are Mutually Independent are picked, such vertices are made simplicial and then removed from the graph. On the remaining set of vertices, Minimum Degree Vertex is applied to make it chordal.Then we take union of these two part of graphs and check for chordality. As soon as graph becomes chordal, we

color vertices using Minimum Vertex Coloring based on the principle that no two adjacent vertices are of the same color. In Chapter 4, Bipartite graphs come into play. Since computing Minimum triangulation is NP-Complete, we use Reduction method to convert Bipartite into Chain graphs via Chordal Graphs adding as few edges as possible. For this purpose we used MDV We used Minimum Degree Vertex on the Bipartite graph also and record the number of edges added. We drew a comparison between the number of edges added via Reduction and MDV. Chapter 5 is about the generation of Nearly Chordal Graphs. We have introduced a heuristic to generate such graphs from Complete graphs by edge deletion and preserving the nearly chordal property.A graph is said to be nearly chordal if all its vertices, have chordal non-neighbors.
The main idea has been to generate triangulations with fewest edges possible. We compared previous methods in terms of edges to support the triangulation problem. The generation of Nearly Chordal also insinuates towards more research in the field. Some of the things to look forward in the future is described in the upcoming section.

6.1 Future Works

The future works has a wide spectrum involving many interesting topics. As we have generated nearly chordal graphs, there should be detailed characterizations of it in place. Once can explore triangulation problem in Nearly Chordal Graphs. The relationship between chordal, nearly chordal and weakly chordal graphs may be studied.
The cataloguing problem can be solved for both chordal and nearly chordal graphs.The cataloguing can be performed can be performed for smaller graphs. The cataloguing problem will generate all chordal or nearly chordal graphs for a particular number of nodes.Such catalogues will be helpful in providing general idea about graphs and their properties.
An open problem could be put forth to find a sequence of vertices for elimination.

Given a chordal graph G, starting with any clique R of G, we can add the remaining vertices one by one, each time adjoining a new vertex to a clique of the graph obtained so far. When all the vertices not in R have been added we get G. The challenge lies in finding such a sequence of vertices in $G - R$ constructively.

BIBLIOGRAPHY

[1] Anne Berry. A wide-range efficient algorithm for minimal triangulation. In *SODA*, volume 99, pages 860–861. Citeseer, 1999.

[2] Jean RS Blair, Pinar Heggernes, and Jan Arne Telle. Making an arbitrary filled graph minimal by removing fill edges. In *Scandinavian Workshop on Algorithm Theory*, pages 173–184. Springer, 1996.

[3] Jean RS Blair, Pinar Heggernes, and Jan Arne Telle. A practical algorithm for making filled graphs minimal. *Theoretical Computer Science*, 250(1-2):125–141, 2001.

[4] Jean RS Blair and Barry Peyton. An introduction to chordal graphs and clique trees. In *Graph theory and sparse matrix computation*, pages 1–29. Springer, 1993.

[5] LM Blumenthal. Theory and applications of. *Distance Geometry*, 1970.

[6] Andreas Brandstädt and Chính T Hoàng. On clique separators, nearly chordal graphs, and the maximum weight stable set problem. *Theoretical Computer Science*, 389(1-2):295–306, 2007.

[7] Peter Buneman et al. A characterisation of rigid circuit graphs. *Discrete mathematics*, 9(3):205–212, 1974.

[8] Gabriel Andrew Dirac. On rigid circuit graphs. In *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, volume 25, pages 71–76. Springer, 1961.

[9] Delbert Fulkerson and Oliver Gross. Incidence matrices and interval graphs. *Pacific journal of mathematics*, 15(3):835–855, 1965.

[10] Fănică Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Series B*, 16(1):47–56, 1974.

[11] GeeksForGeeks. Welsh powell graph colouring, 2019.

[12] Alan George and Joseph WH Liu. The evolution of the minimum degree ordering algorithm. *Siam review*, 31(1):1–19, 1989.

[13] Martin C Golumbic. Algorithmic aspects of perfect graphs. *Annals of Discrete Mathematics*, 21:301–323, 1984.

[14] Ryan B Hayward. Weakly triangulated graphs. *Journal of Combinatorial Theory, Series B*, 39(3):200–208, 1985.

[15] Pinar Heggernes. Minimal triangulations of graphs: A survey. *Discrete Mathematics*, 306(3):297–317, 2006.

[16] Louis Ibarra. Fully dynamic algorithms for chordal graphs. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 923–924, 1999.

[17] Sudiksha Khanduja, Aayushi Srivastava, Asish Mukhopadhyay, and Md. Zamilur Rahman. Generating weakly chordal graphs from arbitrary graphs. *arXiv:2003.13786v1*, 2020.

[18] Lilian Markenzon, Oswaldo Vernet, and Luiz Henrique Araujo. Two methods for the generation of chordal graphs. *Annals of Operations Research*, 157(1):47–60, 2008.

[19] Tatsuo Ohtsuki. A fast algorithm for finding an optimal ordering for vertex elimination on a graph. *SIAM Journal on Computing*, 5(1):133–145, 1976.

[20] Md Zamilur Rahman, Udayamoorthy Navaneetha Krishnan, Cory Jeane, Asish Mukhopadhyay, and Yash P Aneja. A distance matrix completion approach to 1-round algorithms for point placement in the plane. In *Transactions on Computational Science XXXIII*, pages 97–114. Springer, 2018.

[21] Fujis M. Rahman Z., Dhillon T. and Mukhopadhyay A. Generation of k-chromatic chordal graphs, 2017.

[22] Ronald C Read. A survey of graph generation techniques. In *Combinatorial mathematics VIII*, pages 77–89. Springer, 1981.

[23] Donald J Rose, R Endre Tarjan, and George S Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on computing*, 5(2):266–283, 1976.

[24] Oylum Şeker, Pinar Heggernes, Tınaz Ekim, and Z Caner Taşkın. Generation of random chordal graphs using subtrees of a tree. *arXiv preprint arXiv:1810.13326*, 2018.

[25] RE Tarjan. Maximum cardinality search and chordal graphs. 1976. *Unpublished lecture notes for CS*, 259.

[26] Robert E Tarjan and Mihalis Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on computing*, 13(3):566–579, 1984.

[27] Dominic JA Welsh and Martin B Powell. An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal*, 10(1):85–86, 1967.

[28] Mihalis Yannakakis. Computing the minimum fill-in is np-complete. *SIAM Journal on Algebraic Discrete Methods*, 2(1):77–79, 1981.

VITA AUCTORIS

Name: Aayushi Srivastava

Place of Birth: Lucknow, Uttar Pradesh, India

Year: 1993

Education: Bachelor of Technology in Computer Science and Engineering from Amity University, India (2012 - 2016)

Masters in Computer Science at the University of Windsor, Canada (Fall 2018 - Summer 2020)