Impact of Graph Operations on Connectivity

(Combinatorics and Complexity)

A THESIS

submitted by

N.SADAGOPAN

for the award of the degree

of

DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

NOVEMBER 2011

THESIS CERTIFICATE

This is to certify that the thesis titled **Impact of Graph Operations on Connectivity** (Combinatorics and Complexity), submitted by **N.Sadagopan**, to the Indian Institute of Technology, Madras, for the award of the degree of **Doctor of Philosophy**, is a bonafide record of the research work done by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. N.S.Narayanaswamy
Research Guide
Associate Professor
Department of Computer Science and Engineering
Indian Institute of Technology Madras

Place: Chennai

Date:

This thesis is dedicated to

To my Mother **(Mathru Devo Bhava)**

Smt.N.Sundaranayaki

To my Father **(Pithru Devo Bhava)**

Sri.T.Narasimha Iyengar

To my Acharyans **(Acharya Devo Bhava)**

Family Guru: Sri.K.K.Annan Swamy

Thesis Guru: Dr(Sri).N.S.Narayanaswamy

ACKNOWLEDGEMENTS

I would like to convey my humble namaskarams and profound gratitude to my advisor Dr.N.S.Narayanaswamy, as this thesis is as much his as it is mine. I am grateful to him for taking me as his student and guiding me throughout this tenure. He has made significant contributions and has been thoroughly involved in shaping my research career. Without his guidance I would not have been able to do this volume of work. His profound ideas and fundamental discoveries have shaped my thesis to a great extent. His faith in me and hope that I will certainly do good work transformed me to what I am today. His dedication and commitment towards research were beyond comparisons. He has unique way of inculcating research and I consider him to be the most perfect research advisor. I owe a lot to him for all that I know about doing research and for always pointing out interesting and stimulating problems to me. His clarity of thought, careful guidance, constructive feedback, and invaluable suggestions transformed all my reports into research papers.

I am grateful for the amount of time he has devoted to me. During conference deadlines, he adjusted all his schedule for my sake. He sacrificed many weekends before conference deadlines. He even sacrificed his sleep, got up as early as 3 am and worked with me till 12 pm. I admire his patriotism, hard work and undeviating focus which display marks of great research. He also ensured that I emerge as a successful teacher. Apart from his wonderful research guidance, his out-standing teaching had tremendous influence in my professional growth. Everything that I know about theoretical computer science is due to him. Without his blessings, I would not have tasted the flavor of algorithmic graph theory. He is a complete repository of knowledge. He also taught me English. It is because of him that I have written this thesis. An enormous thanks to him, for being understanding and patient, for being a great source of information and insight, and for giving me the freedom to work at my own pace. His patience and expertise in bringing out the best out of a student is beyond appreciations, while his inspiration can lift any student to excellence. I consider myself blessed as my research work was carried out in traditional 'guru-sishya' style.

He was also my philosopher and very good friend indeed. He gave me moral support, taught me values, and instilled confidence in me to face life. Although, words cannot express my gratitude, I thank him once again for all these and many more. His role in my professional growth has been most praiseworthy. He has been a binding force and a catalyst. His wavelength with me is excellent and it is no coincidence that my performance has gone up a notch in the last couple of years. This unique and solid bond should make me remember my advisor not merely on Teacher's day but every moment of my life. The 'Veda' (ancient Hindu philosophy) considers 'gurus' equivalent to God and it is indeed true in my case. My guru can not be compared with anyone but himself. With this happy note I call my advisor a 'living God'. Acharya Devo Bhava! I must have done some good act to some one some time-'Enna punniyam sei teyno'. I wish I could do PDF under his captaincy.

IIT Madras is a temple of knowledge and the teachers are agents of knowledge. I am grateful to all the faculty at the Department of Computer Science and the Department of Mathematics. Everything that I understand about Theoretical Computer Science is due to their creative and inspiring teaching. My deep sense of gratitude to Prof.S.A.Choudum, Prof.Rama, Prof.Arindama Singh, Prof.C.Pandurangan, Prof.Kamala Krithivasan, Prof.P.Sreenivasa Kumar, Prof.Chandrasekar, Prof.T.A.Gonsalves, Prof.Hema A.Murthy, Prof.V.Kamakoti, Dr.N.S.Narayanaswamy, Dr.A.V.Jayanthan, and Dr.Anurag Mittal. I wish to place on record two of my 'all-time' favourite teachers.

First and foremost, I would like to extend my sincere gratitude to Prof.S.A.Choudum who introduced me to the exciting field of graph theory. My interest in structural graph theory was instilled in me by Prof.S.A.Choudum, who taught me many classical theorems. I have great admiration for his teaching. I would not have been able to do this volume of work had it been non-graph theory research. Many thanks to him. In addition, I call him 'a gentle giant of Mathematics'. Second, my humble pranams to Prof.C.Pandurangan for his course on 'joy of algorithms'. 'The intrinsic bhava' (feel/emotive appeal) in his teaching leaves me awe struck every time I listen to him. I greatly benefited from his remarkable lecture series. His scintillating lectures on 'lower bound theory' and 'order statistics' are still green in my mind and inspired many. Highlights of his lectures are; 'amazing' dissemination of his knowledge on the subject, his way of appreciating other's work, beautiful philosophical thoughts and many more. In my view, being a good teacher brings many feathers to one's cap and Prof.CPR is a live example and a great asset to IITM.

It gives me immense pleasure to express my deepest gratitude to Prof.Kamala Krithivasan (a legend in Automata theory), with whom I worked as teaching assistant for the past three years. She was very instrumental in improving my teaching skills. Her expert guidance and constant inspiration motivated me to work with greater zeal and enthusiasm. It is my privilege to have the opportunity to work under her supervision.

I am grateful to Dr.Andrew Thangaraj, Dr.Sounaka Mishra, and Dr.Shailesh Vaya for providing sound constructive criticism during my DC meetings.

During a short but exciting visit to IISc, my discussion with Dr.Sunil, Mathew, Rogers, Anita, and Manu was a rewarding experience. Especially, those long hours of discussion with Mathew, Rogers, and Anita taught me how to get a deep insight into the problem at hand. I am grateful to Dr.Sunil and his team and my association with them was thoroughly enjoyable.

I am indebted to my collaborators, Mrinal and Gaurav of IITM for spending their valuable time. Many thanks to them for enriching my skills in structural graph theory.

It is my pleasure to thank all the faculty at IMSc for their support and unconditional help. In particular, I thank Prof.Venkatesh Raman (A classic teacher), Dr.Saket Saurabh for providing me stimulating and thought provoking lecture series on graph theory and parameterized complexity.

My special thanks to Mr.Parathasarathy (ISRO), R.Shyamsundar (Adobe), Chester (IITKgp), and Dr.Devanathan (TI) with whom I had the opportunity to do coursework. They also spent their valuable time with me for discussing many tutorials and assignments.

My perception of Combinatorics and Complexity in general has changed a lot in the last two years under the influence of my good natured friends. I am pleased to mention Rakesh Mohanty, G.Ramakrishna, Anju Srinivasan, Sajin Koroth, Krithika Ramaswamy, and R.Subhashini. Including me, we form K_7. I express my heartfelt thanks to each one of them for their support and encouragement. I thank Rakesh Mohanty

for being my co-passenger, encouraging me to attend conferences, and providing many valuable suggestions during this journey. Rama, Krithika, and Sajin deserves special appreciations. This smart K_3 provided me intellectual support in the form of reading and correcting my reports. With this intelligent K_3, I had many logical fights while trying to understand the nuances of theory research. I also had the opportunity to enrich my other skills in theory because of their association. I should be grateful to them.
My special thanks to all our interns, in particular, Meera (PSG) and Ramya (MIT) with whom I had the opportunity to share a little that I know and validate my understanding of the subject.
An enormous thanks to N.Mahathi and Smt.Lakshmi Narayanaswamy, my special friends, for their affection, support, and cooperation at different stages of my research. Their style of hospitality is unmatchable.
Besides excellent professors, IITM is endowed with a very efficient and affable office staff as well. Mrs.Radhai and Mrs. Saradha do a spectacular job at administration. I am very grateful to them. I also thank Mr.Balu and Mrs.Prema for maintaining an excellent library in our department. My extreme gratitude to our HOD and Dean for providing me HTRA and also, a huge travel support to attend all my conferences.
AIDB lab has been a great place to work and perfect atmosphere to write proofs. I thank all my seniors, in particular, Dr.Ravindranath chowdary for creating wonderful atmosphere. I also thank all AIDB Lab members and all those who helped me in completing this research work.
Salutations to my family members for their support and encouragement. My humble namaskarams to all my family members. At least a crore namaskarams (it is trivial in words!) to my parents who nurtured in me many good values including 'sathyam vada' (always speak truth) and 'dharmam chara' (help others). Anything good in me is due to them. I am also grateful to my immediate elder brother Sri.N.Alagiamanavalan for his help and support during my coursework and comprehensive examination and most importantly for tuning my mind towards carnatic music. I owe a lot to him.

ABSTRACT

KEYWORDS: *Vertex Connectivity, Edge Contraction, Contractible Edges, Connectivity Augmentation, Connected Vertex Separators*

The objective of this thesis is to study the impact of edge (non-edge) contraction and edge addition on vertex connectivity. An edge (non-edge) in a k-vertex connected graph is *contractible* if its contraction does not result in a graph with lesser vertex connectivity; otherwise the edge (non-edge) is called non-contractible. The following are some of the main results presented in this thesis and we highlight them under three heads.

Contractible Vertex Pairs

[1] A graph is chordal if it does not contain any induced cycle of length at least 4. We present a structural characterization of contractible edges in k-connected chordal graphs. We show that an edge e in a k-connected chordal graph G is contractible if and only if either e is contained in a unique maximal clique of G or e is contained in the intersection of two maximal cliques such that the size of the intersection is strictly more than the vertex connectivity of G.

[2] We show that the graph formed on the set of contractible edges of any k-connected chordal graph ($k \geq 2$) is at least 2-connected.

[3] From the study of non-edges in chordal graphs, we show that each non-edge is contractible. Further, the resultant graph is chordal if and only if there is no induced path of length at least 5 connecting the non-edge.

[4] We present a structural characterization of graphs in which every minimal vertex separator is an independent set. Further, such a graph class has the property that every edge is contractible. We show that a graph G does not have a cycle of length at least 4 with exactly one chord as an induced subgraph if and only if every minimal vertex separator in G is an independent set.

[5] From our study of non-edges restricted to 2-connected graphs, we show that cycles are the only 2-connected graphs in which each non-edge is non-contractible.

Reducing Connectivity to one by Edge Contractions

[1] We present the equivalence between the computational problems of finding a minimum number of edges to be contracted to reduce the (s,t)-vertex connectivity to one and a minimum connected (s,t)-vertex separator.

[2] We study the computational complexity of minimum connected (s,t)-vertex separator ((s,t)-CVS) and show that (s,t)-CVS is $\Omega(log^{2-\epsilon}n)$-hard, for any $\epsilon > 0$, unless NP has quasi-polynomial Las-Vegas algorithms. i.e., for any $\epsilon > 0$ and for some $\delta > 0$, (s,t)-CVS is unlikely to have $\delta.log^{2-\epsilon}n$-approximation algorithm.

[3] By considering chordality as the parameter, we show that (s,t)-CVS is NP-complete on graphs with chordality at least 5 and polynomial-time solvable on bipartite chordality 4 graphs. We also present a $\lceil \frac{c}{2} \rceil$-approximation algorithm for (s,t)-CVS on graphs with chordality c.

[4] Finally, from the parameterized setting, we show that (s,t)-CVS parameterized above the (s,t)-vertex connectivity is $W[2]$-hard.

Connectivity Augmentation

Given a k-vertex connected graph G, vertex connectivity augmentation determines a smallest set of edges whose augmentation to G makes it $(k+1)$-vertex connected. We propose a framework for connectivity augmentation in graphs. Given a connected graph, our framework first constructs the associated tree by understanding the structural decomposition of graphs with respect to its minimum vertex separators. This tree is unique and it captures all minimum vertex separators in the given graph. We focus our combinatorial analysis on the associated tree to decide an optimum connectivity augmentation set. As a first step, we propose an equivalence relation on the set of leaves of the tree. The set of equivalence classes is our data structure, using which we determine the edge to be augmented to the graph at each iteration. Moreover, this partition guarantees a recursive sub-problem and it is sufficient to maintain this partition for finding an optimum connectivity augmentation set in graphs. In this thesis, we report biconnectivity augmentation of 1-connected graphs using biconnected component trees, triconnectivity augmentation of 2-connected graphs using 3-block trees, and $(k+1)$-connectivity augmentation of k-trees and k-connected chordal graphs using minimum vertex separator trees and clique separator trees, respectively. As regards the run-time, chordal augmentation runs in cubic time and other augmentations run in linear time.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

1.1 Graph-theoretic Preliminaries

We follow standard graph theoretic definitions and notation [1, 2]. In this thesis, we consider connected undirected unweighted simple graphs.

- $G = (V, E)$ denotes a connected undirected unweighted simple graph where $V(G)$ is the set of vertices and $E(G) \subseteq \{\{u, v\} \mid u, v \in V(G), u \neq v\}$ is the set of edges. A pair $\{u, v\} \subseteq V(G)$ is said to be adjacent if $\{u, v\} \in E(G)$, otherwise, $\{u, v\}$ is said to be non-adjacent. A non-adjacent pair is called a non-edge. We use n and m to denote $|V(G)|$ and $|E(G)|$, respectively.
- The *neighborhood* of a vertex v in G is the set $N_G(v) = \{u \mid \{u, v\} \in E(G)\}$.
- The *degree* of a vertex v, denoted as $deg_G(v) = |N_G(v)|$. $\delta(G)$ and $\Delta(G)$ denote the *minimum* and *maximum* degree of vertices, respectively in G.
- A *path* P on the vertex set $V(P) = \{u = v_1, v_2, \ldots, v_n = v\}$ (where $n \geq 2$) has its edge set $E(P) = \{\{v_i, v_{i+1}\} \mid 1 \leq i \leq n-1\}$. Such a path is denoted by P_{uv} or $v_1 v_2 \ldots v_n$. Note that any path on 2 vertices is just an edge. The *length* of P is $|V(P)|$. A graph G is connected if for all $u, v \in V(G)$, there exists a path P_{uv}.
- A graph H is a *subgraph* of G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. A subgraph H is an *induced subgraph* if for all $u, v \in V(H)$, $\{u, v\} \in E(H)$ if and only if $\{u, v\} \in E(G)$.
- For $S \subset V(G)$, $G[S]$ denote the induced graph on the set S and $G \setminus S$ is the induced graph on the vertex set $V(G) \setminus S$.
- For a connected non-complete graph G, a *vertex separator* $S \subset V(G)$ is such that $G \setminus S$ has more than one connected component. A vertex separator is minimal if no proper subset of it is a vertex separator. A minimum vertex separator is a minimal vertex separator of least cardinality. The *vertex connectivity* of G, written $\kappa(G)$, is the size of a minimum vertex separator. G is *k-connected* if $\kappa(G) = k$. If a minimal vertex separator is a singleton set then it is called a *cut vertex*. The focus of this thesis is on vertex connectivity related problems and hence, we use k-connected graphs and k-vertex connected graphs interchangeably.
- For a minimal vertex separator S and for some connected component A in $G \setminus S$, the graph G_{A+S} denote the induced subgraph on the vertex set $V(A) \cup S$.

- For a connected non-complete graph G and a non-adjacent pair $\{s,t\}$, a vertex separator $S \subset V(G)$ is called a (s,t)-*vertex separator* if in $G \setminus S$, s and t are in two different connected components and S is minimal if no proper subset of it is a (s,t)-vertex separator. A minimum (s,t)-vertex separator is a minimal (s,t)-vertex separator of least size. The (s,t)-vertex connectivity denote the size of a minimum (s,t)-vertex separator. A connected (s,t)-vertex separator S is a (s,t)-vertex separator such that $G[S]$ is connected and such a set S of least size is a minimum connected (s,t)-vertex separator. For a minimal (s,t)-vertex separator S, let C_s and C_t denote the connected components of $G \setminus S$ such that s is in C_s and t is in C_t. Note that every minimal vertex separator of G is a minimal (s,t)-vertex separator for some non-adjacent pair $\{s,t\}$.
- For a pair $\{u,v\} \subset V(G)$, we let $G \cdot uv$ denote the graph obtained by contracting the pair $\{u,v\}$ in G such that $V(G \cdot uv) = V(G) \setminus \{u,v\} \cup \{z_{uv}\}$ and $E(G \cdot uv) = \{\{z_{uv}, x\} \mid \{u,x\} \text{ or } \{v,x\} \in E(G)\} \cup \{\{x,y\} \mid \{x,y\} \in E(G) \text{ and } x \neq u, y \neq v\}$. If $e = \{u,v\} \in E(G)$ then $G \cdot uv$ is denoted as $G \cdot e$, otherwise, it is $G \cdot uv$ itself.

Definition 1.1.1 *A* clique *in a graph is a set of pairwise adjacent vertices and an* independent set *(or* stable set*) in a graph is a set of pairwise non-adjacent vertices.*

Definition 1.1.2 *A* chordal graph *is a graph with no induced cycle of length at least 4. A graph is said to have* chordality c *if there is no induced cycle of length at least* $c+1$*. Note that chordal graphs have chordality 3.*

Definition 1.1.3 *An edge* $e = \{u,v\}$ *in a graph* G *is* contractible *if* $\kappa(G \cdot e) \geq \kappa(G)$*. Otherwise,* e *is* non-contractible*. Similarly, a non-edge* $\{u,v\}$ *is* contractible *if* $\kappa(G \cdot uv) \geq \kappa(G)$*. Otherwise,* $\{u,v\}$ *is* non-contractible*.*

Definition 1.1.4 *For a* k*-connected graph* G*, a* connectivity augmentation set $E_{min}(G)$ *is a smallest set of edges whose* augmentation *to* G *makes it* $(k+1)$*-connected.*

1.2 Algorithmic Preliminaries

Definition 1.2.1 *Let* $f(n)$ *and* $g(n)$ *denote the running time of two algorithms on inputs of size* n*. We say* $f(n) = O(g(n))$ *(which means that* f *grows no faster than* g*) if there is a constant* $\delta > 0$ *such that* $f(n) \leq \delta.g(n)$*.*

Definition 1.2.2 *The class P denotes the set of problems that can be solved in polynomial time using deterministic algorithms. The class NP denotes the set of problems that can be solved in polynomial time using non-deterministic algorithms.*

Definition 1.2.3 *A problem π is said to be NP-hard if for all $\pi' \in NP$, there exists a polynomial-time reduction from π' to π. π is said to be NP-complete if π is also in NP.*

Definition 1.2.4 *For a minimization problem $\mathcal{P}$, an α-approximation algorithm $\mathcal{A}$ is a polynomial-time algorithm that produces a solution that is no larger than $\alpha.\mathcal{A}^*$ where $\mathcal{A}^*$ is the optimal solution for the problem $\mathcal{P}$. The problem $\mathcal{P}$ is said to admit a α-approximation algorithm.*

Definition 1.2.5 *An algorithm is said to run in Quasi-polynomial time if its worst case running time is $2^{O((logn)^c)}$ for some constant $c > 0$.*

Definition 1.2.6 *A Las-vegas algorithm is a randomized algorithm that always gives correct results. The randomness is not on the output. For example, a randomized quicksort is an example of a Las-vegas algorithm where the pivot is chosen randomly.*

Definition 1.2.7 *An optimization problem $\mathcal{P}$ is $\Omega(f(n))$-hard if there exists a constant $c > 0$ so that $\mathcal{P}$ admits no $c.f(n)$-approximation algorithm, unless NP has quasi-polynomial Las-Vegas algorithms (or P=NP).*

Definition 1.2.8 *A tree-decomposition of a graph G is a tree T where each node $x \in V(T)$ has a label $l(x) \subseteq V(G)$ such that;*

- $\bigcup_{x \in V(T)} l(x) = V(G)$. *(We say that "all vertices are covered.")*
- *For each edge $\{u, v\}$ in G, there exists an $x \in V(T)$ such that $u, v \in l(x)$. (We say that "all edges are covered.")*
- *For any $v \in V(G)$, the nodes $\{x \in V(T) \mid v \in l(x)\}$ form a connected subtree of T. (We call this the "connectivity condition")*

Given a tree-decomposition T, the width of the decomposition is $max_{x \in V(T)}|l(x)| - 1$. The tree width of a graph G is the minimum k such that G has a tree-decomposition of width k. It is easy to see that every graph G has tree width at most $n - 1$, since a single node with all vertices in it is a tree-decomposition for G and has width $n - 1$. In the next chapter, we recall tree-decomposition in the context of chordal graphs and highlight that each node in the tree-decomposition is a maximal clique.
The following lemma is well-known in the literature as *Expansion lemma* [1] and we use it in the next chapter while proving our claims.

Lemma 1.2.1 (Expansion Lemma) *Let G be a connected graph and $G' = G \cup \{v\}$ be the graph obtained from G by adding the vertex v such that $deg_{G'}(v) \geq k$. If $\kappa(G) = k$ then $\kappa(G') \geq k$.*

1.2.1 A Note on Parameterized Complexity

Parameterized complexity theory has been introduced by Downey and Fellows [3] and it is a two dimensional generalization of classical complexity theory (P vs NP). Typically, a parameterized decision problem is defined by specifying the input, the parameter, and the question to be answered. For example, a parameterized decision version of vertex cover problem is formulated as follows: Given a graph G and a parameter k, does there exist a set $S \subset V(G)$ such that $|S| \leq k$ and $G \setminus S$ is an independent set. A parameterized problem is said to be *fixed-parameter tractable* (FPT) if it admits an algorithm whose running time on an instance (x, k) is bounded by $f(k) \cdot |x|^{O(1)}$, where f is an arbitrary function depending only on k, otherwise, it is fixed-parameter intractable. Note that when k is fixed, the problem becomes tractable in the input size x. For example, the parameterized vertex cover has a fixed-parameter tractable algorithm that runs in time $O(2^k \cdot |x|^2)$. The class of fixed-parameter tractable problems is denoted by FPT. In classical complexity, algorithms are analyzed based on the input size n whereas, in parameterized complexity, algorithmic analysis is based on two input parameters, input size n and the parameter k. The secondary measurement called the parameter k also affects the computational complexity of the problem of interest. For more information on this topic, refer the books by Downey and Fellows [3], and Niedermeier [4].

In order to characterize those problems that are unlikely to admit FPT algorithms, Downey and Fellows defined a *parameterized reduction* and a hierarchy of intractable parameterized problem classes above FPT: the popular classes are $W[1]$ and $W[2]$. A parameterized problem A is *reducible* to a parameterized problem B if there is an algorithm ϕ which transforms an instance (x, k) of A into an instance (x', k') of B and such that

1. $(x, k) \in A$ if and only if $(x', k') \in B$
2. k' depends only on k, there exists a function g such that $k' = g(k)$
3. ϕ runs in time $f(k) \cdot |x|^{O(1)}$ for some function f depending only on k.

From the above definition, if a parameterized problem A reduces to B and $B \in W[1]$, then $A \in W[1]$ as well. To prove a parameterized problem A, $W[1]$-complete, we need to establish two facts, A is in $W[1]$ and A is $W[1]$-Hard. To achieve this, choose a $W[1]$-complete problem, say, B, and show a parameterized reduction from B to A and vice versa. The former reduction proves that A is $W[1]$-hard and the latter reduction proves that A is in $W[1]$. We refer [4] for the definition and more examples of parameterized reduction between parameterized problems. The parameterized intractability class $W[1]$ is an analogue of the classical intractability class NP. Several natural parameterized problems are known to be $W[1]$-complete and $W[2]$-complete. The popular ones are parameterized independent set is $W[1]$-complete and parameterized dominating set is $W[2]$-complete.

1.3 Evolution of our Research

Edge contraction and edge addition are two fundamental operations performed on graphs. While doing so, they create an impact on graph parameters such as connectivity, maximum degree, diameter, etc. This thesis explores the impact of graph operations on vertex connectivity. Our initial focus was on finding a structural characterization of edges whose contraction does not decrease the connectivity, such edges are popularly called *contractible edges*. Our contribution as part of research is for a special graph class, *chordal graphs*. We have obtained a structural characterization of contractible

edges in chordal graphs and using which we have analyzed the graph formed on the set of contractible edges. We highlight that there are graphs in which every edge is contractible. An attempt in this direction led to a structural characterization of graphs in which every minimal vertex separator is a stable set. Further, in such graphs every edge is contractible. From our study on non-edges restricted to 2-connected graphs, we have proved that cycles are the only 2-connected graphs in which each non-edge is non-contractible. Our next focus was on analyzing the computational complexity of reducing the (s,t)-vertex connectivity by one using a minimum number of edge contractions. It is important to note that contracting a non-contractible edge decreases the vertex connectivity by one. However, not all graphs have non-contractible edges. In such a case, it is natural to contract a set of edges to reduce the vertex connectivity by one. We consider the related problem: What is the computational complexity of reducing the (s,t)-vertex connectivity to one using a minimum number of edge contractions? We have shown that this problem is computationally equivalent to the problem of finding a minimum connected (s,t)-vertex separator ((s,t)-CVS). Further, we have shown that (s,t)-CVS is NP-complete and hard to approximate within $\delta.\log^{2-\epsilon} n$, for all $\epsilon > 0$ and some $\delta > 0$. By considering chordality as the parameter, we have shown that (s,t)-CVS on chordality 5 graphs is NP-complete and polynomial-time solvable on bipartite chordality 4 graphs. An investigation of (s,t)-CVS from the perspective of parameterized complexity has shown that parameterizing above the (s,t)-vertex connectivity is hard for the complexity class $W[2]$. We have also shown that reducing the (s,t)-vertex connectivity to one is a special instance of reducing the (s,t)-vertex connectivity by one. Therefore, reducing the (s,t)-vertex connectivity by one is also NP-complete. Our final focus was to come up with a unified framework for connectivity augmentation. Connectivity augmentation is a study of the impact of edge additions on vertex connectivity. The objective is to find a minimum number of edges to be augmented to a graph to increase its vertex connectivity by one. Our research contribution towards this goal is an integrated framework using which we have proposed a linear-time algorithm for connectivity augmentation in 1-connected graphs, 2-connected graphs, and k-trees. We have also presented a cubic time algorithm for connectivity augmentation in k-connected chordal graphs. We believe that the above results nicely demonstrate the message of this thesis.

1.3.1 Connections to Practical Applications

A Graph is a mathematical object that models many problems in computer networks and database systems. We highlight a few applications which fall into the scope of this thesis. Connectivity models robustness of a computer network. A network with robustness value k can tolerate at most $(k-1)$ node failures (fault tolerant factor is $(k-1)$).

Contractible vertex pairs map to fault tolerant systems preserving robustness: In a network, a node failure is a common scenario and in such a case the links incident on that node go inactive. Now, requests to this node can be handled by any of its adjacent (non-adjacent) node and that must be accomplished without compromising on the robustness of a network. This can be modeled as edge (non-edge) contraction not decreasing the connectivity. If such a contractible vertex pair exists, then the network can still tolerate $(k-1)$ node failures. By doing so, the network topology may change and the network overhead may increase. However, there is a trade off between robustness and other network parameters.

Connected vertex separators map to graph clustering: A typical database system maintains a collection of documents (nodes) and some relationships (edges) between them. A cluster is a subcollection of similar documents. A database system can be seen as a collection of small clusters. For example, a citation graph. To record the references between clusters there is a set of monitors which keeps track of the number of inter cluster references and other statistical parameters. Every intercluster communication is through one or more monitors and hence, monitors act as a bottleneck or in graph-theoretic term, is a vertex separator. For statistical purpose, we allow monitors to communicate to each other and our goal is to optimize the number of monitors. This can be modeled as an instance of the connected vertex separator problem.

Connectivity augmentation maps to Resilient systems: The objective of connectivity augmentation is to add as few links as possible to a network to increase its robustness (connectivity) by one. By doing so, the fault tolerant factor of the network increases by one and hence, the network is resilient to k failures.

While the problems considered in this thesis have relevance to practical applications, the focus of this thesis is to understand the underlying combinatorics and analyze its complexity.

1.3.2 Organization of the Thesis

The study of contractible edges (non-edges) in chordal graphs is reported in Chapter 2. In Chapter 3, we present the structural characterization of graphs in which every minimal vertex separator is an independent set. The results on non-contractible non-edges restricted to 2-connected graphs is presented in Chapter 4. Complexity results on connected (s, t)-vertex separator are reported in Chapter 5. Finally, in Chapters 6 and 7, we present results on connectivity augmentation in graphs.

CHAPTER 2

Contractible Vertex Pairs in Chordal Graphs

The goal of this chapter is to explore chordal graphs and its structural characterization with respect to minimal vertex separators. Using this characterization, we then present our results on the structure of contractible edges in chordal graphs. Finally, we discuss contractible non-edges in chordal graphs.

2.1 Edge Contraction and Contractible Pairs: A Survey

The aim of this section is to survey results on contractible edges and non-edges in graphs. We begin this section by introducing edge contraction and its applications. *Edge Contraction:* In a graph G, contraction of an edge $e = \{u, v\}$ is the replacement of u and v with a single vertex z. In the resulting graph, denoted by $G \cdot e$, the edges incident on u and v are incident on z. Edge contraction is a very popular operation in Graph Theory and related areas. It is one of the three operations used to define the minor of a graph: H is a *minor* of G if H can be obtained from G by a sequence of edge deletions or edge contractions or isolated vertex deletions. Forbidden minors are used to obtain characterization of many classes of graphs specified by structural properties. A classic example is the forbidden minor characterization of planar graphs. Edge contraction and in general 'clique contraction' plays a significant role in the proof of the Perfect Graph Theorem [2], and in other structural results [5]. Edge contraction also plays a significant role in randomized algorithms for min-cut by exploiting the basic idea that contracting a randomly chosen edge does not increase the size of the min-cut [6]. This leads to expected polynomial-time algorithms for min-cut, and these algorithms are fundamentally different from the classical max-flow based techniques. In practical settings edge contraction is used to model link failures in networks and edge contraction plays an interesting role in the design of reliable networks. Also, edge contraction has applications in many areas in structural graph theory [1], for example, counting the number of

spanning trees. Recall that an edge (non-edge) is contractible if its contraction does not decrease the connectivity.

Contractible Edges: As with many a problem in Graph Theory, the study of contractible edges was initiated by Tutte in [7] where a constructive characterization of 3-connected graphs was presented. Tutte's ingenious development of a theory of 3-connected graphs marked the beginning of the study on contractible edges. One consequence of this characterization is that in a 3-connected graph with at least five vertices, there is at least one contractible edge. In the work by Saito et al. [8], this lower bound was improved to $\frac{|V(G)|}{2}$, and the structure of graphs that have exactly so many contractible edges was studied. Dean, Hemminger and Toft in [9], proved that longest paths in 3-connected graphs contain at least one contractible edge. Subsequently, Dean, Hemminger and Ota in [10] showed that the longest cycle in 3-connected graphs have at least three contractible edges. Contractible edges are also studied in classes of 3-connected graphs with specific properties. It is shown that in a 3-connected graph there is a contractible edge in each longest cycle in two separate results, one considering non-Hamiltonian graphs [11], and the other considering Hamiltonian graphs [12]. However, beyond 3-connected graphs the existence of a contractible edge is not guaranteed due to the existence of contractible critical graphs, which is a class of graphs where every edge contraction decreases the connectivity by one. A k-connected graph without any contractible edge is called a *critically k-connected graph*. This calls for a study of structural characterization of contractible edges in restricted and special graph classes. This study was triggered by Martinov in [13] and he showed that unlike 3-connected graphs, there are 4-connected graphs without any contractible edge. Martinov studied critically 4-connected graphs [14], and obtained a recursive characterization of critically 4-connected graphs [13]. He proved that critically 4-connected graphs are either C_n^2 or the line graphs of the cubic cyclically 4-connected graphs. Structural properties of critically 5-connected graphs have been investigated in [15]. Thomassen [16] has shown that there is a contractible edge in a triangle-free k-connected graph in which the minimum degree is more than $\frac{3k-3}{2}$. In [17], it is shown that there are infinitely many planar contraction critically 4-connected graphs. However, almost all of them are far from being maximal planar, which is also reflected by the claim that every maximal planar graph on at least eight vertices contains at least $\lceil \frac{3n}{4} \rceil$ many 4-contractible edges. Apart

from the past results which study the existence of contractible edges in k-connected graphs and its sub-structures, there are results on the distribution of contractible edges in k-connected graphs [18], and other results which study the covering of contractible edges in 3-connected graphs. Kriesell's survey of contractible edges [5] is an excellent source for many results in this area, and is also the motivation point of our work.

Contractible non-edges: Another natural and useful operation is to contract non-edges which do not lower the connectivity and its importance is highlighted by Kriesell in [19]. Structural characterization of contractible non-edges in general is open. However, the study on special graph classes like bipartite graphs and triangle-free graphs has been dealt in the literature. Since the contraction of an edge in a bipartite graph yields a non-bipartite graph, contracting non-edges is a useful operation in bipartite graphs and is reported in [19]. The natural question of characterizing bipartite graphs with no contractible non-edges is addressed in [19]. It also contains a study of contractible non-edges in triangle-free graphs. The structural characterization of contractible non-edges in 3-connected graphs is presented in [20].

2.2 Structural Properties on Chordal graphs

The objective of this section is to present known results from the literature on chordal graphs and present our observations which highlight the structural properties of chordal graphs. Recall that a chordal graph is a graph in which there is no induced cycle of length at least 4. Chordal graphs are a subclass of a well-known graph family, namely, perfect graphs [2]. They are also known as triangulated graphs and have applications in the study of linear sparse systems, scheduling, and relational database systems. Chordal graphs enjoy many structural properties and we below mention a few of them.

Chordal graph preliminaries: Chordal graphs have a recursive definition as well. A chordal graph is defined recursively as follows: a clique is a chordal graph, if G' is a chordal graph, then the graph $G = G' \cup \{v\}$ such that $N_G(v)$ is a clique in G' is also a chordal graph. A vertex $v \in V(G)$ is *simplicial* if $N_G(v)$ induces a clique. *Simplicial ordering or Perfect Elimination Ordering (PEO)* of a graph G is an ordering

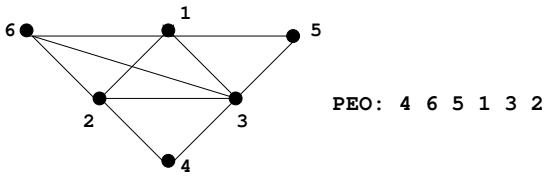


Figure 2.1: A chordal graph and its PEO

$(v_n, \ldots, v_1)$ of $V(G)$ such that for each $1 \leq i \leq n$, v_i is a simplicial vertex in the sub graph induced by $(v_i, \ldots, v_1)$. Chordal graphs admit a very natural PEO which can be viewed as the construction order, i.e. the order in which the vertices were *added into* the chordal graph. An example of a chordal graph and its PEO is illustrated in Figure 2.1. We let $\sigma = (v_n, \ldots, v_1)$ to denote a PEO. $\sigma(x)$ denotes the index of a vertex x in σ. $S_G(x)$ denotes the simplicial neighborhood of x in G, i.e. $\{v \mid v \in N_G(x)$ and $\sigma(x) > \sigma(v)\}$.

2.2.1 Known Results

Theorem 2.2.1 *[2] For a graph G, the following conditions are equivalent.*
(i). G is a chordal graph.
(ii). Every minimal vertex separator in G is a clique.
(iii). There exists a Perfect Elimination Ordering (PEO) of vertices in G.

Theorem 2.2.2 *[2] Let G be a non-complete chordal graph. Then, there exists two non-adjacent simplicial vertices in G. Further, if G is complete, then every vertex in G is simplicial.*

We can represent a chordal graph G using a *tree-like* graph and such a tree is popularly known as *tree-decomposition* tree T of G and it satisfies the following properties;

- For each vertex $x \in V(T)$, the associated label $l(x) \subseteq V(G)$ induces a maximal clique in G.
- For each edge $\{u, v\} \in E(G)$, there exists $x \in V(T)$ such that $\{u, v\} \subseteq l(x)$.
- For each $v \in V(G)$, the subgraph T_v induced by the set $\{x \in V(T) \mid v \in l(x)\}$, is a sub tree in T.

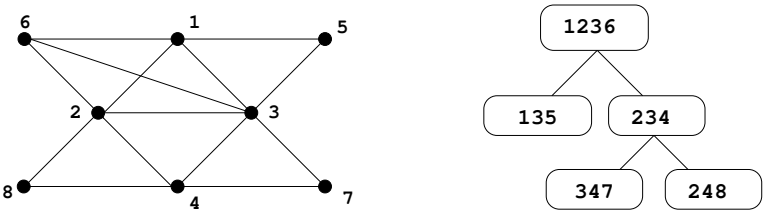


Figure 2.2: A chordal graph and its tree-decomposition tree

An illustration is given in Figure 2.2. Tree-decomposition tree is also known as clique tree [2]. Other representations for chordal graphs include, clique graphs [21] and clique separator graphs [22].

2.2.2 Observations on PEO and Clique trees

We first prove a structural result on k-connected chordal graphs with respect to its PEO. We then present our results on the associated tree decomposition which are subsequently used in proving our structural theorems on contractible edges in chordal graphs.

Lemma 2.2.3 *Let G be a chordal graph with $|V(G)| \geq k+1$ and $\sigma = (v_n, \ldots, v_1)$ be its fixed simplicial ordering. G is k-connected if and only if $\forall v \in V(G), |S_G(v)| \geq k$ and the first $(k+1)$ vertices, i.e. $(v_{k+1}, \ldots, v_1)$ form a $k+1$ clique.*

Proof *Necessity:* Since G is chordal, by Theorem 2.2.2, there exists $\{u, v\} \notin E(G)$ such that u and v are simplicial in G. Since G is k-connected, we claim that $deg_G(u)$ is at least k. For assume, to the contrary, that $deg_G(u)$ is at most $(k-1)$. Clearly, $N_G(u)$ is a vertex separator and hence, G is at most $(k-1)$-connected. Contradicting the assumption that G is k-connected. So, as we claimed, $deg_G(u)$ is at least k. We now claim that $G \setminus \{u\}$ is k-connected. Without loss of generality, we assume that $G \setminus \{u\}$ is a non-complete graph with $|V(G \setminus \{u\})| \geq k+2$. Suppose, on the contrary, $G \setminus \{u\}$ is not k-connected. Clearly, $G \setminus \{u\}$ is chordal and $G \setminus \{u\}$ has a minimum vertex separator S of size $k-1$. Let A and B denote any two connected components in $G \setminus S$. Note that, since $N_G(u)$ is a clique, it follows that either $N_G(u) \subseteq G_{A+S}$ or $N_G(u) \subseteq G_{B+S}$. This implies that S is also a minimum vertex separator in G and hence, G is $(k-1)$-connected. However, by the hypothesis, G is k-connected.

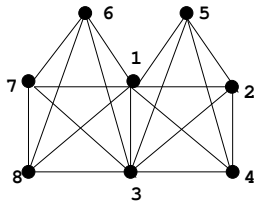


Figure 2.3: An example to show the converse of Lemma 2.2.3 is false

Therefore, $G \setminus \{u\}$ is k-connected. Now, apply Theorem 2.2.2 to get the next vertex in the ordering and so on. Repeat this procedure till $|V(G)|$ becomes a clique of size at least $k+1$. In this case, any ordering of $V(G)$ suffices. *Sufficiency:* We provide a proof by induction on n. For the base case, we consider G with $(k+1)$ vertices. Clearly, by the given fact G is a $(k+1)$ clique and hence, G is k-connected. For the induction hypothesis, assume that G on $(n-1)$ vertices satisfying our premise is k-connected. To complete the induction, let G be a chordal graph with n vertices and $(v_n, \ldots, v_1)$ denote a simplicial ordering of G. Consider the graph $G \setminus \{v_n\}$. Since v_n is simplicial, $G \setminus \{v_n\}$ is connected. In particular, by our induction hypothesis, $G \setminus \{v_n\}$ is k-connected. Since $N_G(v_n) = S_G(v_n)$, which is at least k, by Lemma 1.2.1, it follows that adding v_n to $G \setminus \{v_n\}$, which is G itself is also k-connected. Hence the lemma. □

Corollary 2.2.4 *Let G be a chordal graph. If G is k-connected, then each maximal clique is of size at least $k+1$.*

Proof From Lemma 2.2.3, it follows that for all v, $v \cup S_G(v)$ is a maximal clique of size at least $k+1$. □

The converse of the above claim is not true. An example is given in Figure 2.3.

Lemma 2.2.5 *Let G be a chordal graph and T be its tree decomposition. G is connected if and only if for each edge $\{x, y\} \in E(T)$, $l(x) \cap l(y) \neq \phi$.*

Proof *Necessity:* If G is connected, then we need to show that for each edge $\{x, y\} \in E(T)$, $l(x) \cap l(y) \neq \phi$. We prove this by contradiction. Suppose there exists an edge $\{x, y\} \in E(T)$ and $l(x) \cap l(y) = \phi$. Consider the two components C_1 and C_2 obtained

by removing the edge $\{x, y\}$. Assume that $x \in V(C_1)$ and $y \in V(C_2)$. Let $A = \bigcup_{z \in V(C_1)} l(z)$, $B = \bigcup_{z \in V(C_2)} l(z)$. Since T is a tree decomposition and $l(x) \cap l(y) = \phi$, it follows that $A \cap B = \phi$. Further, each edge $e \in E(G)$ is contained in the graph induced by A or B but not both. So, G is disconnected. However, by our hypothesis G is connected. Hence, our assumption is wrong. Therefore, if G is connected, then for each edge $\{x, y\} \in E(T)$, $l(x) \cap l(y) \neq \phi$.

Sufficiency: Given that for each edge $\{x, y\} \in E(T)$, $l(x) \cap l(y) \neq \phi$, we now show that G is connected. We show that $\forall u, \forall v \in V(G), u \neq v$, there exists a path between u and v in G. Let x, y be any two vertices in $V(T)$ such that $u \in l(x)$ and $v \in l(y)$. Consider the path $x = z_1, z_2, ..., z_j = y$ in the tree T. Further, $l(z_i) \cap l(z_{i+1})$ is non empty in T. This implies that there exists a vertex $r_i \in l(z_i) \cap l(z_{i+1})$. Hence, the sequence of edges $\{u, r_1\}\{r_1, r_2\}...\{r_{j-1}, v\}$ is a uv path in G. The reason this is true in G, is because G is a chordal graph, and the label of each node in T is a maximal clique. Therefore, u and v are connected in G. Hence, G is connected. □

Theorem 2.2.6 *Let G be a k-connected chordal graph and let T be its tree decomposition. Let M be the set of minimal vertex separators of G. Let $M' = \{X \mid X = l(x) \cap l(y)$ where $\{x, y\} \in E(T)\}$ and $M'' = \{Y \mid Y \in M'$ and for all $Z \in M', Z \not\subset Y\}$. $M = M''$. In other words, M'' is the set of minimal vertex separators of G.*

Proof $M'' \subseteq M$:

Let $S \in M''$. Clearly, for some $\{x, y\} \in E(T)$, $S = l(x) \cap l(y)$. By definition, $T \setminus S$ is a tree decomposition of $G \setminus S$ and $l(x) \cap l(y) = \phi$. From Lemma 2.2.5, it follows that $G \setminus S$ is disconnected. Hence, S is a vertex separator. Further, since $S \in M''$, it follows that there is no $S' \subset S$ such that $G \setminus S'$ is disconnected. Therefore, S is a minimal vertex separator.

$M \subseteq M''$:

Let S be a minimal vertex separator (MVS). We now show that $S \in M''$. We now argue that there exist distinct $x, y \in V(T)$ such that $\{x, y\} \in E(T)$ and $l(x) \cap l(y) = S$. Since S is a MVS, $G \setminus S$ is disconnected. Since $G \setminus S$ is disconnected, in $T \setminus S$ there exists a pair of vertices x and y such that $l(x) \cap l(y) = \phi$. We now claim that in T, $S = l(x) \cap l(y)$. If $l(x) \cap l(y) \subset S$, this implies that S is not a MVS. However, we are

given the fact that S is a MVS. Therefore, $S = l(x) \cap l(y)$ and consequently, $S \in M'$. Since S is a MVS, it follows that $S \in M''$. Hence the proof. $\square$

2.3 Structure of Contractible Edges in Chordal Graphs

In Theorem 2.3.1, we establish the necessary and sufficient condition for an edge to be contractible in a k-connected chordal graph.

Theorem 2.3.1 *Let G be a k-connected chordal graph with $|V(G)| \geq (k+2)$. An edge $e = \{u, v\} \in E(G)$ is contractible if and only if one of the following holds*
(i) e is in a unique maximal clique in G.
(ii) For $x, y \in V(T)$, $\{u, v\} \subset l(x) \cap l(y)$ such that $\{x, y\} \in E(T)$ and $|l(x) \cap l(y)| > k$.

Proof *Necessity:* We prove this claim by contradiction. Suppose, e is neither in a unique maximal clique in G nor e is contained in two maximal cliques such that their intersection is at least $k + 1$. This implies, e is in two maximal cliques and for some $\{x, y\} \in E(T)$, $\{u, v\} \subset l(x) \cap l(y)$ and $|l(x) \cap l(y)| \leq k$. On contraction of e in G, the tree decomposition of $G \cdot e$ is $T.e$. In $T.e$, the $|l(x) \cap l(y)| \leq k - 1$. From Lemma 2.2.5, it follows that $l(x) \cap l(y)$ is a vertex separator of $G \cdot e$, and since $|l(x) \cap l(y)| \leq k - 1$, it follows that $G \cdot e$ is at most $(k - 1)$-connected. However, given that e is contractible implies that $G \cdot e$ is k-connected. Therefore, our assumption is wrong. Hence, the necessity. *Sufficiency:* First, we consider the case when e is in a unique maximal clique and show that e is contractible. If e is in a unique maximal clique in G, then it implies that e is contained in the label of a unique node in T.
Case 1: $|V(T.e)| = |V(T)|$. In $T.e$, for each $x, y \in T$, $|l(x) \cap l(y)|$ remains unchanged in $T.e$. From Theorem 2.2.6, the connectivity of $G \cdot e$ is same as the connectivity of G. Therefore, e is contractible.
Case 2: $|V(T.e)| = |V(T)| - 1$. In other words, e is contained in a node z of T and in $T.e$, the node z no longer exists. Therefore, in $T.e$, for each $\{x, y\} \subseteq V(T)$, $z \notin \{x, y\}$, $|l(x) \cap l(y)|$ remains unchanged in $T.e$. From Theorem 2.2.6, the connectivity of $G \cdot e$ is at least as much as the connectivity of G. Therefore, e is contractible. In the case, when $|l(x) \cap l(y)| > k$ for all $\{x, y\} \in E(T)$, after contracting e, in $T.e$, $|l(x) \cap l(y)|$

is at least k and hence, by Theorem 2.2.6, the connectivity of $G \cdot e$ is at least k. Hence, $G \cdot e$ is k-connected. Therefore, e is contractible in G. □

Corollary 2.3.2 *If for each $z \in V(G)$, $e = \{u, v\} \not\subset S_G(z)$, then $e = \{u, v\}$ is contractible.*

Proof Since for any $z \in V(G)$, $\{u, v\} \not\subset S_G(z)$, implies that e is in a unique maximal clique. By Theorem 2.3.1, e is contractible. □

2.3.1 Bound on the Number of Contractible Edges

As a consequence of Theorems 2.2.2 and 2.3.1, it follows that all edges incident on simplicial vertices are contractible. Therefore, any non-complete k-connected chordal graph has at least $2k$ contractible edges. This bound is tight, since any non-complete k-connected chordal graph on $k + 2$ vertices has this bound. We claim that, any k-connected chordal graph on n vertices has at most $\binom{n-1}{2} + k - \binom{k}{2}$ contractible edges. We prove this claim by induction on n. For the base case, consider any k-connected chordal graph on $(k+1)$ vertices. Assume the claim is true for any k-connected chordal graph on $n - 1$ vertices ($n \geq k + 2$). Consider a k-connected chordal graph G on n vertices. Let v be a simplicial vertex in G. Consider the graph $G \setminus \{v\}$. If $G \setminus \{v\}$ is k-connected, then by the induction hypothesis there are at most $\binom{n-2}{2} + k - \binom{k}{2}$ contractible edges in $G \setminus \{v\}$. Now introduce v into $G \setminus \{v\}$. Clearly the introduction of v can create at most $n - 1$ new contractible edges. i.e. $\binom{n-2}{2} + k - \binom{k}{2} + n - 1$ which is $\binom{n-1}{2} + k - \binom{k}{2}$. If $\kappa(G \setminus \{v\}) > k$, then every edge in $G \setminus \{v\}$ is contractible. Therefore, the number of contractible edges in G is $\binom{n-2}{2} + k - \binom{k}{2} < \binom{n-1}{2} + k - \binom{k}{2}$. This bound is also tight as it is evident from the following construction: consider any k-connected chordal graph obtained from a complete graph on $n - 1$ vertices by adding a simplicial vertex of degree k.

2.4 Distribution of Contractible Edges in Chordal Graphs

Given a k-connected chordal graph G, we now investigate the structure of the graph formed by the set $E_c(G)$ of contractible edges. We denote such a graph by G_c. i.e. $V(G_c) = V(G)$ and $E(G_c) = E_c(G)$. For $k = 1$, each edge in G is contractible, therefore, G_c is G itself. For $k \geq 2$, we show that G_c is at least 2-connected.

Theorem 2.4.1 *For $k \geq 2$, let G be a non-complete k-connected chordal graph. G_c is at least 2-connected.*

Proof We prove by induction on the number of vertices of G.

Base: Let G be any non-complete k-connected chordal graph on $(k+2)$ vertices. G_c has exactly two vertices u and v of degree k such that both u and v are adjacent to an independent set of size k. G_c is clearly 2-connected.

Hypothesis: Let G be any non-complete k-connected chordal graph on $(n-1)$ vertices. We assume that our theorem is true in G.

Induction Step: Let G be a non-complete k-connected chordal graph on n vertices. Consider a simplicial vertex w in G (By Theorem 2.2.2, such a vertex exists in G). Consider the graph $G \setminus \{w\}$.

Case 1: $G \setminus \{w\}$ is a complete graph on $(n-1)$-vertices. Note that $\kappa(G \setminus \{w\}) \geq k$. Since G is non-complete, $|N_G(w)| < n-1$. In other words, there exists w' in $G \setminus \{w\}$ such that $\{w, w'\} \notin E(G)$. Now, the set of edges $\{\{w, x\} \mid x \in N_G(w)\}$ are contractible as per Theorem 2.3.1. Clearly, the set of contractible edges $\{\{z, x\} \mid z \in \{w, w'\}, x \in N_G(w)\}$ form a 2-connected graph. If there is a vertex $w''(\neq w')$ in $G \setminus \{w\}$ then it is clearly adjacent to at least two vertices in G_c and hence, by Lemma 1.2.1, G_c is at least 2-connected. By a similar argument, when $G \setminus \{w\}$ is non-complete and $\kappa(G \setminus \{w\}) > k$, we can show that G_c is at least 2-connected.

Case 2: $G \setminus \{w\}$ is a non-complete chordal graph on $(n-1)$ vertices with $\kappa(G \setminus \{w\}) = k$. By the induction hypothesis $(G \setminus \{w\})_c$ is at least 2-connected. We now prove that G_c is at least 2-connected when w is introduced into $G \setminus \{w\}$. We prove by contradiction that G_c is at least 2-connected. Assume that G_c is not at least 2-connected. This implies that there exists a cut vertex z in G_c. Without loss of generality, we assume that $G_c \setminus \{z\}$ has two components. Let C_1 and C_2 be two components in $G_c \setminus \{z\}$. Let

$[N_G(w)] = \{w\} \cup N_G(w)$. Note that both $[N_G(w)] \cap V(C_1)$ and $[N_G(w)] \cap V(C_2)$ can not be non-empty. The reason this is true is due to the fact that w is simplicial and all edges incident on w are in a unique maximal clique and hence, by Theorem 2.3.1, all edges incident on w are contractible. Hence, $[N_G(w)] \cap V(C_1) \neq \phi$ and $[N_G(w)] \cap V(C_2) = \phi$. Consider the induced subgraph of $(G \setminus \{w\})_c$ induced by the set $V(C_1) \setminus \{w\}$. Clearly, $V(C_1 \setminus \{w\}) \cap V(C_2) = \phi$. This implies that $E(C_1 \setminus \{w\}) \cap E(C_2) = \phi$. Also, observe that C_2 is connected. Consider the set $A = \{e \mid e = \{u, v\}, u \in V(C_1 \setminus \{w\})$ and $v \in V(C_2)\}$. We claim that the set A is empty. Suppose A is not empty, then there exists an edge $e = \{u, v\} \in A$. Clearly, $e \notin G_c$. Therefore, $\{u, v\} \subset N_G(w)$. Hence, $\{u, v\} \subset V(C_1)$. This contradicts the definition of the set A. It follows that the set A is empty. Now, removing z from $(G \setminus \{w\})_c$ increases the number of components in $(G \setminus \{w\})_c$. This implies that z is a cut vertex. Therefore, $(G \setminus \{w\})_c$ is not at least 2-connected. Contradicting the hypothesis and hence, our assumption that G_c is not at least 2-connected is wrong. Therefore, G_c is at least 2-connected. □

Corollary 2.4.2 *Let G be a k-connected chordal graph. Every vertex v in G has at least two contractible edges incident on it.*

Proof From Theorem 2.4.1, G_c is at least 2-connected graph, and hence, $\delta(G_c) \geq 2$. Hence the corollary. □

2.5 Contractible Non-Edges in Chordal Graphs

The focus of this section is on the structural results concerning contractible non-edges in chordal graphs. We first show that contracting any non-edge does not decrease the connectivity and then we characterize those non-edges whose contraction preserves the chordality as well.

Theorem 2.5.1 *Let G be a k-connected chordal graph. For all $\{u, v\} \notin E(G)$, $G \cdot uv$ is k-connected.*

Proof We present a proof by contradiction. Assume, on the contrary, $G \cdot uv$ is not k-connected. This implies that there exists a minimum vertex separator S of size at most

$k-1$ containing z_{uv}. Now uncontract the vertex z_{uv}. Clearly, S is a vertex separator in G and $|S| \leq k$. If S is not minimal, then there exists $S' \subset S$ such that $G \setminus S'$ is disconnected. Note that $|S'| \leq k-1$ and hence, G is at most $(k-1)$-connected. A contradiction to the given fact that G is k-connected. If S is minimal then $G[S]$ is not a clique as $\{u, v\} \notin E(G)$. However, by Theorem 2.2.1, $G[S]$ must be a clique. We again arrive at a contradiction. Therefore, the claim $G \cdot uv$ is k-connected follows. □

Theorem 2.5.2 *Let G be a k-connected chordal graph and $\{u, v\} \notin E(G)$. $G \cdot uv$ is chordal if and only if there is no induced path of length at least 5 connecting u and v in G.*

Proof Let P_{uv} denote an induced path of length at least 5 connecting u and v in G. *Necessity:* Suppose there exists a path P_{uv} in G. This implies that in $G \cdot uv$, there is an induced cycle of length at least 4 and hence, $G \cdot uv$ is not chordal by definition. A contradiction. Hence, the necessity. *Sufficiency:* Given that there is no P_{uv}. For assume, on the contrary, $G \cdot uv$ is not chordal. This implies that there exists an induced cycle C of length at least 4 in $G \cdot uv$. Clearly, C contains the vertex z_{uv} created due to the contraction of the non-edge $\{u, v\}$. If not, then we get a contradiction to the fact that G is chordal. Let $(v_1, v_2, ..., v_k = z_{uv}, v_1), k \geq 4$ denote an arbitrary ordering of vertices in C. We consider two observations to complete the proof. First, since C is an induced cycle, it follows that for v_i, v_j in C such that $1 \leq i \neq j \leq k$ and $|i-j| > 1$, $\{v_i, v_j\} \notin E(G)$. From the above argument, we see that there exists P_{uv} of length at least 5 in G. This contradicts our hypothesis that there is no such path. Second, it may be the case that $G \cdot uv$ is not chordal and $\{u, v_1\}, \{v, v_{k-1}\} \in E(G)$. In this case, we see that $\{v_1, \ldots, v_{k-1}, u\}$ induces a cycle of length at least 4 in G. However, G is chordal. A contradiction. This shows that $\{u, v_1\}, \{v, v_{k-1}\} \notin E(G)$. Again, we see that there exists P_{uv} of length at least 5 in G. In either case, we contradict our hypothesis that there is no such path. Therefore, $G \cdot uv$ is chordal. Hence, the theorem. □

Bound on the number of contractible non-edges: Let $\mathcal{S}$ be the set of simplicial vertices in a chordal graph and $s = |\mathcal{S}|$. Since all simplicial vertices in any chordal graph are non-adjacent, as per Theorem 2.5.1, every pair of vertices in S is contractible. Hence, a lower bound on the number of contractible non-edges is given by $|E_{nc}| \geq$

$\frac{s(s-1)}{2}$. Consider the following argument to compute an upper bound on the number of contractible non-edges. Since a graph on n vertices can have at most $\frac{n(n-1)}{2}$ edges, the number of non-edges can be at most $\frac{n(n-1)}{2} - |E(G)|$. For a connected graph, $|E(G)| \geq n-1$. Therefore, the number of non-edges is at most $\frac{(n-1)(n-2)}{2}$. By Theorem 2.5.1, we know that every non-edge in any chordal graph is contractible. Therefore, $|E_{nc}| \leq \frac{(n-1)(n-2)}{2}$. Any star graph on n vertices has this bound and hence it is tight.

Discussion: We pose out of combinatorial curiosity the following two structural questions. Characterize a graph class such that every non-edge is contractible. Similarly, characterize a graph class in which every edge is contractible. Observe that graphs with each non-edge contractible must have the property that each of its minimum vertex separators induces a clique. So, characterizing graphs with each non-edge contractible is equivalent to characterizing graphs in which each minimum vertex separator induces a clique. From a structural-view, it is appropriate to characterize graphs in which every minimal vertex separator is a clique. Although, it only characterizes a subclass of the graph class of interest, this is the best that we can get from a structural-view. Note that it is precisely the class of chordal graphs. We address in the next chapter, an analogous problem of characterizing graphs in which every edge is contractible. Equivalently, characterize graphs in which every minimum vertex separator is an independent set. Here again, we focus our attention on characterizing graphs in which every minimal vertex separator is an independent set.

Concluding Remarks: This chapter shines a light on structural results on chordal graphs with respect to its tree decomposition and minimal vertex separators. Using this, we have established a necessary and sufficient condition for an edge to be contractible in chordal graphs. Also, considered the related problem of characterizing contractible non-edges in chordal graphs and proved structural results on contractible non-edges in k-connected chordal graphs. We have investigated the structure of the graph formed by the contractible edges in chordal graphs and showed that it is at least 2-connected. Finally, we have highlighted a lower and an upper bound on the number of contractible edges and non-edges in chordal graphs.

CHAPTER 3

Stable Separator Graphs: Any Edge is Contractible

The goal of this chapter is to characterize graph classes based on the properties of minimal vertex (edge) separators. We first present a structural characterization of graphs in which every minimal vertex separator is a stable set (independent set). Moreover, such a graph class has the property that each edge is contractible. We present a structural characterization of such graphs using forbidden graphs and we also analyze the complexity of deciding a maximum such forbidden subgraph.

3.1 Motivation

We present two motivations: one from the theory of constrained vertex separators and other from the theory of contractible edges. Vertex connectivity is a classical topic in graph theory and motivated many problems in structural graph theory. One such problem is the study of constrained vertex separators. In particular, clique separators, stable separators, and balanced separators are the most popularly studied constrained vertex separator in the literature [23, 24, 25, 26]. This line of study was initiated by Dirac [23] with a structural characterization of clique vertex separators graphs. In particular, Dirac addressed a fundamental question of characterizing graphs in which every minimal vertex separator is a clique. In [23], Dirac proved that a graph is chordal if and only if every minimal vertex separator is a clique. Subsequently, chordal graphs were studied in great detail due to its application both in structural and algorithmic graph theory. While chordal graphs and its structural properties have received much attention in the literature, the analogous question: characterize graphs such that every minimal vertex separator is an independent set has not received attention in the past.

Another motivation for the study of stable minimal vertex separator graphs is from the theory of *contractible edges*. Note that in trees every edge is contractible. An immediate question is the following: Is this observation true in graphs as well? If not, what

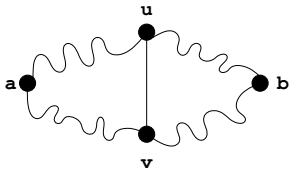


Figure 3.1: A 1-chord graph

is the structure of graphs in which every edge is contractible. Observe that an edge is contractible if it is not contained in any minimum vertex separator. In the light of this observation, we pose a natural question, which is to characterize graphs such that every edge of it is contractible. Equivalently, characterize graphs in which every minimum vertex separator is a stable set. However, from a structural-view, it is appropriate to characterize graphs in which every minimal vertex separator is a stable set.

3.2 Stable Vertex Separator graphs: A Structural Characterization

In this section, we characterize graphs in which all minimal (a, b)-vertex separators are stable sets. For simplicity, we use minimal vertex separator instead of minimal (a, b)-vertex separator and the pair (a, b) will be clear from the context.

Definition 3.2.1 *A* chord *in a cycle is an edge joining a pair of non-adjacent vertices. A subgraph H of G is said to be* 1-chord *if $G[V(H)]$ is a cycle of length at least 4 with exactly one chord. G is* 1-chord free *if G contains no such H. An example is shown in Figure 3.1.*

Lemma 3.2.1 *Let S be a (a, b)-vertex separator in G and C_a and C_b are the connected components in $G \setminus S$ such that $a \in V(C_a)$ and $b \in V(C_b)$. S is minimal if and only if for each $v \in S$, there exists $u \in V(C_a)$ and there exists $u' \in V(C_b)$ such that $\{u, v\}, \{u', v\} \in E(G)$.*

Proof Suppose there exists $v \in S$, for all $u \in V(C_a)$, there is no edge $\{u, v\}$ in G. Consider the set $S' = S \setminus \{v\}$. Clearly in $G \setminus S'$, the pair $\{a, b\}$ are in distinct

components. This implies that S' is a (a, b)-vertex separator. Since $S' \subset S$, implies that S is not a minimal (a, b)-vertex separator. This contradicts the given fact that S is a minimal (a, b)-vertex separator. Therefore, for each $v \in S$, there exists $u \in V(C_a)$ such that $\{u, v\} \in E(G)$. A similar argument proves that for each $v \in S$, there exists $u \in V(C_b)$ such that $\{u, v\} \in E(G)$. Conversely, assume there exists $S' \subset S$ in G such that S' is a minimal (a, b)-vertex separator. Let C'_a and C'_b are the connected components in $G \setminus S'$ such that $a \in V(C'_a)$ and $b \in V(C'_b)$. This implies that for each $z \in V(C'_a)$ and for each $z' \in V(C'_b)$ every path $P_{zz'}$ from z to z' contains an element from S'. Since $S' \subset S$ there must exist $w \in S \setminus S'$ such that $\{w, w'\} \notin E(G)$ for any $w' \in V(C_a)$ or $\{w, w''\} \notin E(G)$ for any $w'' \in V(C_b)$. However, this contradicts the given hypothesis that for each $v \in S$, there exists $u \in V(C_a)$ and there exists $u' \in V(C_b)$ such that $\{u, v\}, \{u', v\} \in E(G)$. Therefore, the sufficiency follows. Hence the lemma. □

Theorem 3.2.2 *G is 1-chord free if and only if every minimal (a, b)-vertex separator of G is an independent set.*

Proof *Necessity:* Given that G is 1-chord free we now show that every minimal (a, b)-vertex separator is an independent set. We present a proof by contradiction. Suppose there exists a minimal (a, b)-vertex separator S such that $G[S]$ is not an independent set. This implies that there exists $u, v \in S$ such that $\{u, v\} \in E(G)$. Since S is a minimal (a, b)-vertex separator, by Lemma 3.2.1, for each $w \in S$, there exists $z \in V(C_a)$ such that $\{w, z\} \in E(G)$. Similarly, there exists $z' \in V(C_b)$ such that $\{w, z'\} \in E(G)$. Consider shortest paths $P_1 = \{u, z_1, \ldots, z_k, v\}, k \geq 1, z_i \in V(C_a)$ and $P_2 = \{u, w_1, \ldots, w_l, v\}, l \geq 1, w_i \in V(C_b)$. Since P_1 and P_2 are shortest paths between u and v and $\{u, v\} \in E(G)$ implies that P_1 and P_2 together form a cycle of length at least 4 with exactly one chord. In other words, G contains an induced 1-chord graph as a subgraph. However, this is a contradiction to the given hypothesis. Therefore, our assumption that there exists a minimal (a, b)-vertex separator S such that $G[S]$ is not an independent set is wrong. Hence the necessity follows.

Sufficiency: We prove that G is 1-chord free by the method of contradiction. Assume that in G there exists an induced 1-chord graph H as a subgraph. We now construct a minimal (a, b)-vertex separator S such that $G[S]$ is not an independent set. Let $V(H) = \{u, a, z_1, \ldots, z_l = v, y_1, \ldots, y_k = b\}, l \geq 1, k \geq 1$ and $E(H) =$

$\{\{u,a\},\{a,z_1\},\{v,y_1\},\{b,u\},\{u,v\}\} \cup \{\{z_i,z_{i+1}\}, 1 \leq i \leq l-1\} \cup \{\{y_j,y_{j+1}\}, 1 \leq j \leq k-1\}$. Let $X = \{a\} \cup \{z_i \mid z_i \in V(H)\} \setminus \{v\}$ and $Y = \{b\} \cup \{y_j \mid y_j \in V(H)\}$. Note that any minimal vertex separator S in G separating X and Y must contain u and v. Moreover, we observe that S is also a minimal (a,b)-vertex separator in G. The reason this is true is due to the following: clearly, $u \in S$ and if $v \notin S$ then there exists a path between a and b through v, contradicting the fact that S is a vertex separator in G. Therefore, S is a minimal (a,b)-vertex separator such that $\{u,v\} \subset S$. Since $\{u,v\} \in E(G)$ implies that $G[S]$ is not an independent set. A contradiction to the hypothesis. Therefore, G is 1-chord free. Hence the theorem. □

Remark: It is important to compare our work with the results of [27]. In [27], a structure theorem for graphs having no cycles with a unique chord is studied. Also, it is mentioned in [27] that such a graph class can be recognized in polynomial time. However, in [27], a structural characterization with respect to minimal vertex separators has not been studied. We now present two more combinatorial observations on 1-chord free graphs with respect to its vertex connectivity.

Lemma 3.2.3 *Let G be a non-complete at least 2-connected graph. If G is 1-chord free, then G is triangle free.*

Proof Suppose G is not triangle free. Let $\{a,b,c\}$ induce a triangle in G. Since G is at least 2-connected and non-complete, there must exist a path between b and c avoiding a. Let P_{bc} denote a shortest such path and $V(P_{bc}) = \{b, z_1, \ldots, z_i, c\}, i \geq 1$. We now show that G contains a 1-chord graph as a subgraph by considering three cases. An illustration is given in Figure 3.2.

Case 1: There is no edge $\{a, z_j\}, 1 \leq j \leq i$, for any $z_j \in V(P_{bc})$. Clearly, $\{a,b,c\}$ together with P_{bc} induce a 1-chord subgraph in G. However, we know that G is 1-chord free. A contradiction. Therefore, G is triangle free. Note that for other two cases, the value of i is at least 2 as G is a non-complete graph.

Case 2: There exists an edge $\{a, z_j\}, 1 \leq j \leq i-1$, for some $z_j \in V(P_{bc})$. If a is adjacent to more than one z_j, then without loss of generality, we choose the z_j such that j is the least. Now, the set $\{c, a, b, z_1, \ldots, z_j\}$ induces a 1-chord subgraph with $\{a,b\}$ as the chord. A contradiction in this case too.

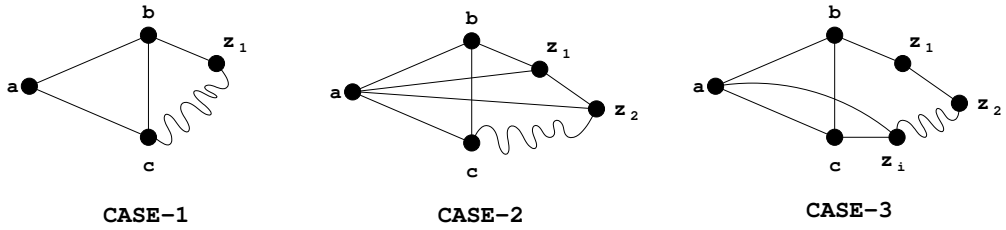


Figure 3.2: An illustration for the proof of Lemma 3.2.3

Case 3: $\{a, z_j\} \in E(G), z_j = z_i$. In this case, the set $\{b, a, c, z_i\}$ is a 1-chord subgraph with $\{a, c\}$ as the chord. A contradiction. This completes our case analysis and hence our assumption that G is not triangle free is wrong. Therefore, the claim follows. □

Note that the converse of the above lemma is not true. For example, trees are triangle free and 1-chord free, however, trees are not 2-connected. The following observation characterizes the connected components in 1-connected 1-chord free graphs. If a property of a graph holds good for each of its induced subgraphs then the property is called a hereditary property. For example, triangle free property.

Lemma 3.2.4 *Let G be an exactly 1-connected 1-chord free graph. For a cut-vertex v of G, let $\{C_1, \ldots, C_r\}$ denote the connected components in $G \setminus \{v\}$. For each cut-vertex v, the subgraph induced on $V(C_i) \cup \{v\}$ is either a complete graph or a 1-chord free subgraph.*

Proof The claim follows from the contrapositive of Lemma 3.2.3 and the fact that 1-chord freeness is a hereditary property. □

3.3 Maximum 1-chord subgraph problem is NP-complete

The characterization in Theorem 3.2.2 can be used to test whether a given graph has the property that every minimal vertex separator is an independent set. In particular, this calls for testing the existence of a 1-chord subgraph in the given graph. Two of the closely related problems are finding a minimum (maximum) 1-chord subgraph of a graph. In the next section, we show that the decision version of a maximum 1-chord

subgraph is NP-complete by establishing a polynomial-time reduction from the Maximum induced cycle problem. The decision version of the maximum 1-chord subgraph is given below.

Maximum 1-chord subgraph problem

Instance: Graph G, and an integer l
Question: Is there a subgraph H of G such that H is 1-chord and $\|V(H)\| \geq l$?

Theorem 3.3.1 *The decision version of maximum 1-chord subgraph of a graph is NP-complete*

Proof 1-chord subgraph is in NP: Given a certificate $C = (G, H, l)$, to witness the fact that this problem is in NP, we now present a deterministic polynomial-time algorithm to verify the validity of C. Observe that a 1-chord graph of size l has the degree sequence $(3, 3, 2, \ldots, 2)$ with exactly $l - 2$ vertices of degree 2. Also, both degree 3 vertices are adjacent and deleting the corresponding edge between them results in a cycle. It is now clear that the above two crucial observations along with standard Depth First Search algorithm, can verify whether C is valid or not, in time polynomial in the input size. Therefore, we conclude that 1-chord subgraph problem is in NP.

1-chord subgraph is NP-hard: We establish a polynomial-time reduction from maximum induced cycle problem and its decision version is known to be NP-complete[28]. The decision version of the problem is given below:

Maximum induced cycle problem

Instance: Graph G, and an integer l
Question: Is there a subgraph H such that H is a cycle and $\|V(H)\| \geq l$?

Given an instance (G, H, k) of induced cycle problem, we construct an instance $(G', H', 2k)$ of 1-chord subgraph as follows: $V(G') = V(G) \cup \{\{v_{ij}^1, v_{ij}^2, \ldots, v_{ij}^k\} \mid \{v_i, v_j\} \in E(G)\}$ and $E(G') = E(G) \cup \{\{v_{ij}^p, v_{ij}^{p+1}\} \mid 1 \leq p \leq k-1\} \cup \{\{v_i, v_{ij}^1\}, \{v_{ij}^k, v_j\}\}$. An example is illustrated in Figure 3.3. We now show that (G, H, k) has an induced cycle of size at least k if and only if $(G', H', 2k)$ has a 1-chord subgraph of size at least $2k$. For *only if* claim, G contains an induced cycle H, $|V(H)| \geq k$. By our construction of G', for any edge $e = \{v_i, v_j\} \in E(H)$, there is a path $P_{v_i v_j}$ using

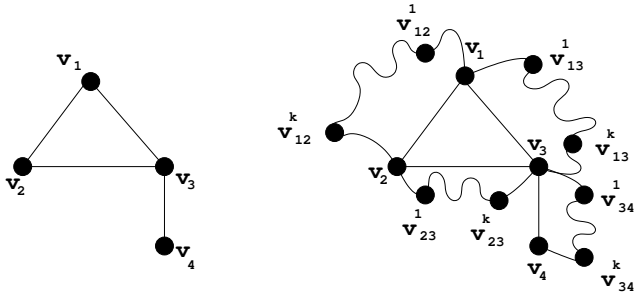


Figure 3.3: Reducing an instance of the induced cycle problem to an instance of the 1-chord graph problem

the vertex set $\{v_{ij}^1, v_{ij}^2, \ldots, v_{ij}^k\}$. Clearly, in G', $V(H)$ together with $\{v_{ij}^1, v_{ij}^2, \ldots, v_{ij}^k\}$ induce a 1-chord subgraph with e as the unique chord. Thus, we have constructed in G', a 1-chord subgraph H' such that $|V(H')| \geq 2k$. For *if* claim, G' contains a 1-chord subgraph H' of size at least $2k$. Note that, if such a H' exists in G', then either $V(H') \cap \{v_{ij}^1, v_{ij}^2, \ldots, v_{ij}^k\} = \phi$ or $\{v_{ij}^1, v_{ij}^2, \ldots, v_{ij}^k\} \subset V(H')$. Also, H' has two induced cycles with at least one is of size at least k. This implies that, in either case, there exists an induced cycle H in G such that $|V(H)| \geq k$. Hence the claim. Note that $|V(G')| = |V(G)| + k|E(G)|$ and $|E(G')| = (k+2)|E(G)|$ and G' can be constructed in $O(k|E(G)|)$ time, which is $O(|E(G)|^2)$. Thus, we have established a polynomial-time reduction from the induced cycle problem to the 1-chord subgraph problem. As a consequence, we conclude that deciding a 1-chord subgraph is NP-hard. Therefore, 1-chord subgraph problem is NP-complete. Hence the theorem. □

Remark: It is interesting to compare our NP-hardness result with an open problem posed in [29] and it is the following: can we determine the longest cycle without crossing chords in polynomial time? Note that maximum induced cycle problem and maximum 1-chord subgraph are special cases of this problem and both are known to be NP-complete. With these observations, we believe that the above problem may not have a polynomial-time algorithm.

Other Observations: As far as approximation algorithm is concerned for maximum 1-chord subgraph problem, there is no polynomial-time approximation algorithm with approximation ratio $O(n^{1-\epsilon})$ for any $\epsilon > 0$. This is true due to the following observation. NP-hard reduction of induced cycle problem is from independent set problem

and this reduction is an approximation ratio preserving reduction. Since independent set does not have an approximation algorithm with approximation ratio $O(n^{1-\epsilon})$ for any $\epsilon > 0$, it follows that maximum induced cycle problem does not have an approximation algorithm with approximation ratio $O(n^{1-\epsilon})$ for any $\epsilon > 0$. Moreover, NP-hard reduction of Theorem 3.3.1 is also an approximation ratio preserving reduction and hence we conclude that 1-chord subgraph problem does not have an approximation algorithm with approximation ratio $O(n^{1-\epsilon})$ for any $\epsilon > 0$. As far as parameterized complexity is concerned with parameter as the size of 1-chord subgraph, we observe that parameterized 1-chord subgraph is $W[1]$-hard. i.e. there is no parameterized algorithm with parameter as the size of 1-chord subgraph. The above result follows from the fact that parameterized independent set is $W[1]$-hard and NP-hard reduction of independent set to induced cycle and NP-hard reduction of Theorem 3.3.1 are parameterized reductions. Therefore, parameterized 1-chord subgraph is $W[1]$-hard.

3.4 A Characterization of Matching Edge Separators

We now focus on the analogous question, which is to characterize the graph class such that any minimal edge separator induces a matching. An edge separator of a graph G is a set $E' \subseteq E(G)$ such that $G' = G \setminus E'$, $V(G') = V(G)$ and $E(G') = E(G) \setminus E'$, has two connected components. An edge separator E' is called a (a, b)-edge separator if E' disconnects a and b. i.e. the graph $G \setminus E'$ has two connected components such that a and b are in distinct components. A (a, b)-edge separator is a minimal (a, b)-edge separator if no proper subset of it is a (a, b)-edge separator.

Theorem 3.4.1 *A graph G is such that for all $a, b \in V(G)$, all minimal (a, b)-edge separators induce a matching if and only if G is a tree*

Proof For *if claim*, since G is a tree, it is a well-known fact that between any pair (a, b) of vertices there exists exactly one path between a and b. This implies that any minimal (a, b)-edge separator contains exactly one edge, which is a matching. Hence the claim. For *only if claim*, we present a proof by the method of contradiction. Suppose G is not a tree. This implies that there exists a cycle C in G. Now, let us consider two vertices

a and b in C such that $\{a, b\} \notin E(G)$. It is a well-known fact that for any two vertices in C there exists two edge disjoint paths between them. In particular, this observation is true for a and b. Let $\mathcal{P} = \{P_{ab} \mid P_{ab}$ is a path between a and b not containing a as an internal vertex $\}$ and $X = \{x \mid x \in V(P_{ab}), P_{ab} \in \mathcal{P}\}$. In other words, the set X is the set of vertices which lie on at least one path P_{ab}. Consider the set $Y = N_G(a) \cap X$, that is Y is the set of all vertices which are adjacent to a and lie on a path from a to b which does not contain a as an internal vertex. Now we will prove that the set of edges $E' = \{\{a, x\} \mid x \in Y\}$ form a minimal (a, b)-edge separator in G. Since a and b lie on C it is clear that E' must contain the edges $\{a, y\}$ and $\{a, z\}$ where $y, z \in V(C)$. This is true due to the fact that $\{a, y\}$ and $\{a, z\}$ lie on two edge disjoint paths between a and b. What follows is that $|E'| \geq 2$ and E' is not a matching, because any two edges in E' has a as the common vertex. We now show that E' is a (a, b)-edge separator. Suppose E' is not a (a, b)-edge separator, then there is a path P'_{ab} from a to b in G not containing any edge from E'. Without loss of generality we assume that P'_{ab} does not contain a as an internal vertex. If P'_{ab} contains a as an internal vertex then we can remove all the vertices until the last a in the path and still get a path from a to b. Let u be the vertex adjacent to a in P'_{ab}, then clearly $u \in X$ and since $\{a, u\} \in E(G)$, $u \in Y$ as well. This implies $\{a, u\} \in E'$, which is a contradiction to our assumption that P'_{ab} does not contain any edges from E'. Therefore, E' is a (a, b)-edge separator. Now to prove the minimality of E', suppose E' is not minimal, then there is at least one edge $e = \{a, x\} \in E'$ such that $E' \setminus \{e\}$ is still a (a, b)-edge separator in G. Since $e \in E'$, we know that $x \in X$ and there is at least one path $P_{ab} \in \mathcal{P}$, containing x as an internal vertex. Now consider the sub path P' of P_{ab} from x to b. It is clear that $a \notin P'$ as P_{ab} by definition does not contain a as an internal vertex. So, there does not exist $y \in V(G)$ such that $\{a, y\} \in E(G)$ and $\{a, y\}$ is an edge of the path P'. In what follows, the edge $\{a, x\}$ together with the path P' yields a path from a to b, which does not contain any edge from $E' \setminus \{e\}$, giving rise to a contradiction that $E \setminus \{e\}$ is a (a, b)-edge separator in G. Hence, we conclude that E' is a minimal (a, b)-edge separator which is not a matching. A contradiction to the hypothesis. Therefore, our assumption that G is not a tree is wrong. Hence the theorem. $\square$

Concluding Remarks: This chapter explores the structural characterization of a graph class in which every edge is contractible. In an attempt to answer this question, we characterized the graph class in which every minimal vertex separator is an independent set. We showed that graph G does not have a cycle of length at least 4 with exactly one chord as an induced subgraph (1-chord subgraph) if and only if every minimal vertex separator in G is a stable set. We also established the fact that deciding whether a graph has a maximum 1-chord subgraph is NP-complete. We also looked at an analogous question in the edge connectivity setting. We showed that the class of graphs in which every minimal edge separator induces a matching are precisely the class of trees. In the next chapter, we restrict our study to 2-connected graphs and characterize graphs such that every non-edge is non-contractible.

CHAPTER 4

Non-Contractible Non-Edges in 2-connected Graphs

The goal of this chapter is to characterize 2-connected graphs in which each non-edge is non-contractible. We show that cycles are the only 2-connected graphs in which each non-edge is non-contractible.

Preliminaries: Notation and Definitions

All graphs considered in this chapter are exactly 2-connected. Recall that a path P on the vertex set $V(P) = \{v_1, v_2, \ldots, v_n\}$ (where $n \geq 2$) has its edge set $E(P) = \{\{v_i, v_{i+1}\} \mid 1 \leq i \leq n-1\}$. Such a path is denoted by $v_1 v_2 \ldots v_n$. If $P = v_1 v_2 \ldots v_n$, then $\dot{P}$ is the path $P - \{v_1, v_n\}$. Vertices v_1 and v_n are said to be the end points of P. If $v_i, v_j \in V(P)$, $v_i P v_j$ is the path $v_i v_{i+1} \ldots v_j$ and $\dot{v_i} P \dot{v_j}$ is the path $v_i P v_j - \{v_i, v_j\}$.

Definition 4.0.1 *Given a subgraph H of G, a path $P = u_1 u_2 \ldots u_m$ in G is said to be an H-path if $V(P) \cap V(H) = \{u_1, u_m\}$ and $E(P) \cap E(H) = \phi$.*

Definition 4.0.2 *Let C be a cycle of length at least 4 in G and $u, v \in V(C)$ such that $\{u, v\} \notin E(G)$. Let P_1 and P_2 be the two paths between u and v in C. A C-path P with end points u' and v' is said to be* across *$\{u, v\}$ if $u', v' \notin \{u, v\}$ and $|\{u', v'\} \cap V(P_1)| = 1$ and $|\{u', v'\} \cap V(P_2)| = 1$, i.e., both u' and v' do not together belong to P_1 or P_2.*

Note that the two components of $C - \{u, v\}$ are $\dot{u} P_1 \dot{v}$ and $\dot{u} P_2 \dot{v}$ and that one of these components contains u' and the other v'. Given two graphs G_1 and G_2, the graph $G' = G_1 \cup G_2$ is such that $V(G') = V(G_1) \cup V(G_2)$ and $E(G') = E(G_1) \cup E(G_2)$.

4.1 Non-edges in 2-connected Graphs

Let G be a 2-connected graph which is not a cycle. Let C be a cycle of length at least 4 in G. Note that a non-edge $\{u, v\}$ is contractible if and only if u and v are not together part of any minimum vertex separator.

Lemma 4.1.1 *If $\{u, v\} \subset V(C)$, $\{u, v\} \notin E(G)$ and there is a C-path P across $\{u, v\}$ with end points u' and v' then there exists $\{x, y\} \notin E(G)$ such that $\{x, y\}$ is a contractible non-edge.*

Proof If $\{u, v\}$ is not a minimum vertex separator then clearly $G \cdot uv$ is 2-connected. Therefore, $\{u, v\}$ is a contractible non-edge. Suppose $\{u, v\}$ is a minimum vertex separator then the graph $G' = G - \{u, v\}$ is disconnected. This implies that G' has at least two components (note that $|G| \geq 4$), say C_1 and C_2. Since P is a C-path across $\{u, v\}$, $u, v \notin V(P)$ and therefore, P is a path from u' to v' in G'. Thus, both u' and v' will be in the same component, say C_1, of G'. $C - \{u, v\}$ has two components, one containing u' and the other containing v', by definition of the C-path P. Since v' is reachable from u' in G', all the vertices of C other than u and v are also reachable from u' in G'. Therefore, $V(C) \setminus \{u, v\} \subseteq C_1$. Now let $w \in C_2$. Clearly, $w \notin V(C)$. Since $\{u, v\}$ is a minimum vertex separator, every path from $u' \in C_1$ to $w \in C_2$ in G must go through $\{u, v\}$. We now claim that $\{u', w\}$ is a contractible non-edge in G. Clearly, $\{u', w\}$ is a non-edge. Let us assume for the sake of contradiction that $\{u', w\}$ is a non-contractible non-edge in G. This means that $\{u', w\}$ is a minimum vertex separator. Let C_3 and C_4 be the two components of $G'' = G - \{u', w\}$. Since $w \notin V(C)$, u and v are connected by a path in G'' and this implies that u and v are in the same component of G'', say C_3. Since $\{u', w\}$ is a minimum vertex separator of G, both u' and w have at least one edge from C_4 incident on them. Note that we now have a path in G between u' and w avoiding $\{u, v\}$ (i.e., through C_4). This contradicts our earlier observation that every path from u' to w in G should pass through u or v. Therefore, $\{u', w\}$ is a contractible non-edge in G. Hence the lemma. □

Lemma 4.1.2 *If $\{u, v\} \subset V(C)$, $\{u, v\} \notin E(G)$ and there is no C-path across $\{u, v\}$, then there exists $\{x, y\} \notin E(G)$ such that $\{x, y\}$ is a contractible non-edge.*

Proof Let P_1 and P_2 be the two paths between u and v in C. Since G is not a cycle, $G \neq C$ and therefore, there exists some C-path P in G. Let u' and v' be the two end points of P. Since P is not across $\{u, v\}$, u' and v' belong to one of P_1 or P_2, say P_1 (note that u' and v' could be u or v). We consider the following two cases:
Case 1: If $||P|| = 1$ (that is, $P = u'v'$) or if $\{u', v'\} = \{u, v\}$: Note that if $P = u'v'$,

then u' and v' are not adjacent in P_1. Therefore, $\dot{u}'P_1\dot{v}'$ is nonempty. If $\{u', v'\} = \{u, v\}$, then $\dot{u}'P_1\dot{v}' = \dot{P}_1$ is again nonempty. Choose vertices $z \in \dot{u}'P_1\dot{v}'$ and $z' \in \dot{P}_2$. See Figure 4.1. If $\{z, z'\} \in E(G)$, then zz' is a C-path across $\{u, v\}$ which contradicts our assumption that there is no C-path across $\{u, v\}$. Therefore, $\{z, z'\} \notin E(G)$. Now, P is a C-path across $\{z, z'\}$ and by Lemma 4.1.1, there exists $\{x, y\} \notin E(G)$ such that $\{x, y\}$ is a contractible non-edge.

Case 2: If $||P|| \geq 2$ and $\{u', v'\} \neq \{u, v\}$: We know that one of u' and v' does

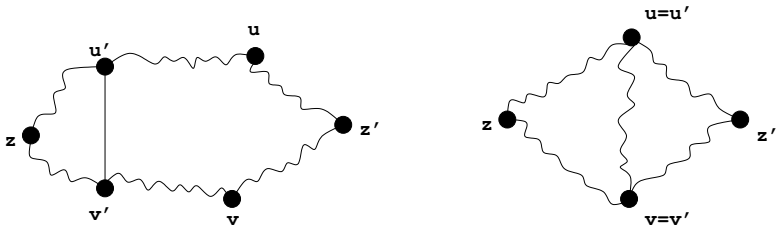


Figure 4.1: An Illustration for Case-1 of Lemma 4.1.2

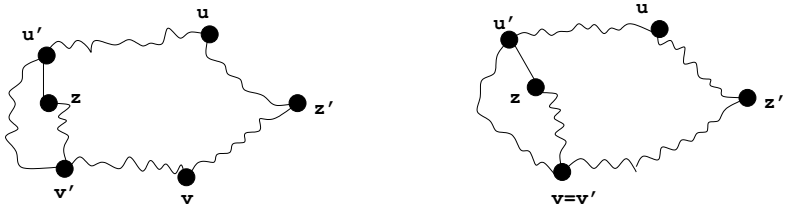


Figure 4.2: An Illustration for Case-2 of Lemma 4.1.2

not belong to $\{u, v\}$. Let us assume without loss of generality that $u' \notin \{u, v\}$. Let C' be the cycle $uP_1u' \cup P \cup v'P_1v \cup P_2$. Pick $z \in \dot{P}$ and $z' \in \dot{P}_2$. See Figure 4.2. If $\{z, z'\} \in E(G)$, then $u'Pzz'$ is a C-path across $\{u, v\}$, which is a contradiction. Therefore, $\{z, z'\} \notin E(G)$. Now, looking at the cycle C', $u'P_1v'$ is a C'-path across $\{z, z'\}$ and therefore by Lemma 4.1.1, there exists $\{x, y\} \notin E(G)$ such that $\{x, y\}$ is a contractible non-edge. □

Lemma 4.1.3 *If G is exactly 2-connected and not a cycle and each non-edge is non-contractible then for all u and v in G, $\{u, v\} \in E(G)$.*

Proof Let us assume that there are two vertices $u, v \in V(G)$ such that $\{u, v\} \notin E(G)$. Since G is 2-connected, by Menger's Theorem, there exists a cycle C of length at

least 4 that contains both u and v. Now, by Lemmas 4.1.1 and 4.1.2, there exists a contractible non-edge in G, which is a contradiction. Therefore, for all $u, v \in V(G)$, $\{u, v\} \in E(G)$. $\square$

Theorem 4.1.4 *G is exactly 2-connected with each non-edge non-contractible if and only if G is a cycle.*

Proof It is easy to see that if G is a cycle, then it is exactly 2-connected and each non-edge is non-contractible. Now if G is exactly 2-connected with each non-edge non-contractible, then by Lemma 4.1.3, either G is a cycle or G is a complete graph. But if G is a complete graph, then G is a triangle since G is exactly 2-connected. Thus, G is, in any case, a cycle. $\square$

4.2 Ear Decomposition and Non-edges

Proposition 4.2.1 *[30] Let H be any cycle which is a subgraph of a graph G. G is 2-connected if and only if it can be constructed from this cycle by successively adding H-paths (ears) to graphs H already constructed.*

Using Proposition 4.2.1, we present an alternate proof for Theorem 4.1.4.

Alternate proof of Theorem 4.1.4

Proof Note that when G is a cycle, its every non-edge is non-contractible. Now suppose G is not a cycle. Then we show that G is not exactly 2-connected. Let C be the smallest cycle in G and let $H = C$. Note that $|V(C)| = |V(H)| \geq 3$. Using Proposition 4.2.1, we can construct G from cycle C by successively adding H-paths H_1, H_2, ..., H_k (in that order) to graphs H already constructed. Let $\mathcal{H} = \{H_1, \cdots, H_k\}$. Observe that since C is the smallest cycle in G, none of the H-paths H_i is a chord of C. For any H-path $H_i = v_1, v_2, \cdots, v_m$ added to an already constructed graph H, we know that $V(H) \cap V(H_i) = \{v_1, v_m\}$. We call the remaining vertices of H_i, i.e. all vertices v_j, where $1 < j < m$, as the *internal* vertices of H_i. Let $V_{int}(H_i)$ denote the set of all internal vertices of H_i. We claim that for any vertex $u \in V(C)$ and $v \in V_{int}(H_i)$,

where $1 \leq i \leq k$, $\{u, v\} \in E(G)$. Let us prove this by contradiction. Assume that the claim is not true. We define $A = \{H_i \in \mathcal{H} \mid u \in V(C), v \in V_{int}(H_i), \{u, v\} \notin E(G)\}$. Let $j = \max_{H_i \in A}\{i\}$. From the definition of A, we know that there exists an $x \in C$ and a $y \in V_{int}(H_j)$ such that $\{x, y\} \notin E(G)$. Let G' denote the graph induced on the vertex set $V' = \bigcup_{1 \leq i \leq j} V(H_i) \cup V(C)$ of G. Now we split the proof of the claim into two cases. In the first case, when $x \notin V(H_j)$, it is easy to see that removing vertices x and y from G' will not disconnect it. Since every internal vertex of any H_i, where $j < i \leq k$, i.e. every vertex $w \in V(G) \setminus V(G')$ has an edge with every vertex of C, removing x and y will not disconnect G too. But this contradicts the fact that every non-edge in G is non-contractible. Hence $\{x, y\} \in E(G)$. i.e., every internal vertex of H_j has an edge with every vertex u of C, where $u \notin V(H_j)$. It also implies that, in the second case when $x \in V(H_j)$, removing x and y from G' will not disconnect it. Now proceeding in the same way as in the first case, we can prove that $\{x, y\} \in E(G)$. This proves the claim. Now from the above claim, it is easy to see that such a graph G is always 3-connected. This contradicts the fact that G is strictly 2-connected. Hence the theorem. □

Note: The number of non-contractible non-edges in any cycle on n vertices is $\binom{n}{2} - n = \frac{n(n-3)}{2}$. We know that, of all 2-connected graphs on n vertices, cycles have the minimum number of edges (equal to n). This implies that, given a 2-connected graph on n vertices, the number of non-contractible non-edges in it is upper bounded by the expression $\frac{n(n-3)}{2}$.

Concluding Remarks: We characterized 2-connected graphs whose every non-edge is non-contractible by showing that only cycles belong to this category. Also, we found a tight upper bound of $\frac{n(n-3)}{2}$ on the number of non-contractible non-edges in any 2-connected graph on n vertices. Note that cycles are a class of graphs in which every edge is contractible and every non-edge is non-contractible. It would be interesting to study other graphs that have this property.

We know that an edge contraction can reduce vertex connectivity due to the existence of non-contractible edges. So, there is an interplay between edge contraction and vertex connectivity. In the next chapter, we focus on the computational problem of reducing the vertex connectivity to one by minimum number of edge contractions.

CHAPTER 5

Reducing Connectivity to one using Edge Contractions

The theme of this chapter is to establish the equivalence between the two computational problems; finding a minimum number of edges to be contracted to reduce the (s,t)-vertex connectivity to one and a minimum connected (s,t)-vertex separator. We also present two motivations: one from the theory of graph minors and the other from the theory of constrained vertex separators. We then analyze the computational complexity of connected (s,t)-vertex separator. Our complexity analysis includes both classical and parameterized view.

5.1 A Motivation from the Theory of Graph Minors

Edge contraction and edge deletion are two important graph-theoretic operations that play a vital role in the study of many graph-theoretic problems. To name a few, counting the number of spanning trees [1], the structural characterization of planar graphs [30], etc. As far as connectivity related problems are concerned, edge deletion is studied extensively with respect to edge connectivity, whereas edge contraction is studied extensively with respect to vertex connectivity. Given an undirected graph and a pair (s,t) of non-adjacent vertices, a minimum (s,t)-edge cut is a minimum set of edges whose removal disconnects s from t. It is well-known that finding a minimum (s,t)-edge cut is equivalent to finding a maximum flow from s to t and obtaining one such (s,t)-edge cut can be done in polynomial time [30, 31]. This approach is popularly used to find the edge connectivity of graphs. The following lemma relates minimum (s,t)-edge cut and edge connectivity.

Lemma 5.1.1 *[30] Let G be a graph with (s,t)-edge connectivity k and $G-e$ denote the graph obtained from G by deleting e. An edge e is not contained in any minimum (s,t)-edge cut of G if and only if the (s,t)-edge connectivity of $G-e$ is k.*

The above lemma implies that deleting an edge in any minimum (s,t)-edge cut reduces the (s,t)-edge connectivity by one. Also, to reduce the (s,t)-edge connectivity to zero, i.e., to disconnect s and t it is enough to find a minimum (s,t)-edge cut. This shows that the complexity of reducing the (s,t)-edge connectivity to zero by minimum number of edge deletions is polynomial time. While edge deletion and its impact on edge connectivity are well-studied using max-flow techniques and polynomial-time algorithms are known, the following lemma relates vertex connectivity and edge contraction.

Lemma 5.1.2 *[30] Let G be a graph with (s,t)-vertex connectivity k and $G \cdot e$ be the graph obtained by contracting the edge e in G. The (s,t)-vertex connectivity of $G \cdot e$ is k if and only if e is not contained in any minimum (s,t)-vertex separator of G.*

In the light of this lemma, we pose the natural analog: Given an undirected graph G and a pair of non-adjacent vertices s and t, find a minimum number of edges to be contracted to reduce the (s,t)-vertex connectivity to one. Note that any edge contraction can not disconnect the graph, it can only reduce the vertex connectivity by at most one. Essentially, edge contraction can be seen as a primitive operation in reducing the (s,t)-vertex connectivity to one. We analyze the complexity of this problem, given the fact that the related problem of reducing the (s,t)-edge connectivity to zero by edge deletion is polynomial-time solvable.

An Important Observation: In this thesis, we observe that there is an equivalence between the computational problems of reducing the (s,t)-vertex connectivity to one using a minimum number of edge contractions and a minimum connected (s,t)-vertex separator. Prior to the analysis of finding a minimum connected (s,t)-vertex separator, we review the past results on constrained vertex separators.

5.2 Constrained Vertex Separators: A Review

The vertex connectivity of a graph and the corresponding separators are of fundamental interest in Computer Science and Graph Theory, and many flavours of vertex separators have attracted researchers in the past. The fundamental problem in the theory of vertex separators is to disconnect a pair (s,t) of vertices by removing a minimum number of

vertices and it is popularly known as minimum (s,t)-vertex cut problem. Interestingly, Menger [1] observed the equivalence between this problem and the maximum number of vertex disjoint paths between s and t. Moreover, a minimum (s,t)-vertex cut is equal to a maximum flow between s and t. Since classical flow theory gives us a way of solving maximum flow between s and t in polynomial time, it follows that minimum (s,t)-vertex cut is polynomial-time solvable. This observation marked the beginning of the theory of separators. In particular, constrained vertex separators and many generalizations of minimum (s,t)-vertex cut have attracted researchers in the past. The first one in this series is the minimum node multiway cut problem: given a graph G and a set $\{t_1,\ldots,t_l\}$ of l terminals, the goal is to find a minimum set S of vertices such that in $G \setminus S$ no two t_i's are in the same component. Note that, when $l = 2$ it is the minimum (s,t)-vertex cut problem. In [32], it was shown that if $l \geq 3$, then node multiway cut is NP-hard. In [32], the more general problem, namely, minimum node multicut was also considered; given a set $\{(s_1,t_1),\ldots,(s_l,t_l)\}$ of l pairs, the objective is to find a minimum set $S \subset V(G)$ such that in $G \setminus S$ no two pairs are in the same component. Observe that slightly changing the definition of a well-understood minimum (s,t)-cut problem usually makes the problem NP-hard. The next variant is α-vertex separator (also known as α-balanced vertex separator) problem. An α-vertex separator is a set $S \subset V(G)$ such that each connected component in $G \setminus S$ has at most $\alpha.n$ vertices. Our goal is to find a minimum α-vertex separator for some constant $\frac{1}{2} \leq \alpha < 1$. For example, any path has a $\frac{1}{2}$-vertex separator consisting of a single vertex; any binary tree has a $\frac{2}{3}$-vertex separator consisting of a single vertex; and some 2-trees have a $\frac{2}{3}$-vertex separator consisting of two vertices. It is important to note that any planar graph has a $\frac{2}{3}$-vertex separator. This result is popularly known as planar-separator theorem, due to Lipton and Tarjan [33]. The NP-hardness result of α-vertex separator was established by Bui and John in [34]. The analogous problem in the edge connectivity setting is also NP-hard as showed in [34]. Marx in [35] investigated three more variants: cutting l vertices, cutting l connected components, and cutting into l components. The goal in cutting l vertices problem is to delete a minimum number of vertices so that exactly l vertices are cut away from the graph. Cutting l connected vertices asks for a minimum number of vertices whose removal leaves a graph in which there is a connected component with exactly l vertices. The objective in cutting into l components problem is

to delete a minimum number of vertices such that the remaining graph has at least l connected components. In [35], the investigation was from parameterized-view, leading to various parameterized hardness results. Hence, these problems are also NP-hard. The edge connectivity version of these problems do not have fixed-parameter tractable algorithms, unless P=NP and were reported in [36]. Essentially, the above results can be seen as imposing constraints on the connected components created due to removal of a separator.

The related problem of imposing a constraint on the vertex separator itself has received considerable attention in the past. Whitesides in [24], initiated the study of finding clique separators in graphs and in [24] it is shown that finding a clique separator is polynomial-time solvable. In [25], Brandstadt et al. have shown that deciding a stable vertex separator is NP-hard. It is NP-hard even on k_4-free graphs.

Interestingly, all of the above problems have received much attention in parameterized complexity as well. Marx initiated this study on node multiway problem and presented in [35], a fixed-parameter tractable algorithm with run time $O(4^{k^3}n^5)$. Later Chen et al. in [37] improved the run time to $O(k4^k n^3)$. Tractable and intractable results from parameterized-view for α-vertex separator and cutting l vertices and its variants are reported in [26]. As far as constrained vertex separators are concerned, Marx in [26] considered the parameterized complexity of constrained separators satisfying some hereditary properties. For example, stable separators. It is shown in [26] that the above problem has an algorithm whose running time is $f(k).n^{O(1)}$, where k is the size of a constrained separator. i.e. these problems are fixed-parameter tractable with parameter as the solution size.

We next mention two noteworthy results in the design of approximation algorithms. Garg et al. designed a $(2 - \frac{2}{l})$-approximation algorithm for node multiway cut problem [38]. In [34], it is shown that α-vertex separator does not have an approximation algorithm with approximation ratio $n^{\frac{1}{2}-\epsilon}$ for any $\epsilon > 0$, unless P=NP.

Besides theoretical importance, all these problems are of practical interest and are investigated as 'graph-partitioning' problems in VLSI design, parallel supercomputing, image processing, communication networks, and in data mining as 'graph-clustering' problems [39, 40, 41, 42].

An associated problem of finding the vertex connectivity of graphs and finding a mini-

mum vertex separator of graphs are well-studied in the literature. The popular approach is to compute the following: for each pair of vertices, find the maximum number of vertex disjoint paths using the max-flow technique. The minimum value over all pairs is the vertex connectivity. There are many algorithms in the literature [43, 44] to compute the vertex connectivity and the fastest in the sequence is due to Gabow [45]. These algorithms also find the corresponding set of vertices of some minimum vertex separator. A related problem of enumerating all minimum or minimal vertex separators of an undirected graph is also studied [46, 47].

Research Gap: While many constrained vertex separators have attracted researchers from both classical and parameterized complexity, the related problem of finding a minimum connected (s,t)-vertex separator is open. In light of [26], this question can also be looked at as finding a (s,t)-vertex separator satisfying some non-hereditary property. For example, connectedness. Moreover, the results in [26] do not carry over to connected (s,t)-vertex separator and its complexity status remains open. In this chapter, we focus our attention on the computational complexity of minimum connected (s,t)-vertex separator ((s,t)-CVS).

5.2.1 Our Contributions

Let G be an undirected connected graph and (s,t) denote a fixed non-adjacent pair of vertices in G. Throughout this chapter, when we refer to edge contraction, we do not contract edges incident on s and edges incident on t.

1. We establish a polynomial-time reduction from the Group Steiner Tree problem [ND12, see [28]] to (s,t)-CVS. Consequently, it follows that there is no polynomial-time approximation algorithm with approximation factor $\delta.log^{2-\epsilon}n$, for some $\delta > 0$ and for any $\epsilon > 0$, unless NP has quasi-polynomial Las-Vegas algorithms.

2. We then observe that on chordal graphs, finding a minimum (s,t)-CVS is polynomial-time solvable as every minimal vertex separator is a clique. With chordality as the parameter, we show that (s,t)-CVS is NP-complete on chordality 5 graphs and polynomial-time solvable on bipartite chordality 4 graphs. We also present a $\lceil \frac{c}{2} \rceil$-approximation algorithm for (s,t)-CVS on graphs with chordality c.

3. We then consider designing algorithms for (s,t)-CVS whose running time is $f(k).n^{O(1)}$

where k is the parameter of interest and f is a function independent of n. If the parameter of interest is the chordality c of the graph, then it follows from the above result that (s,t)-CVS is unlikely to have algorithm whose running time is $f(c).n^{O(1)}, c \geq 5$, unless P=NP. Whereas, on graphs of treewidth ω, we show the existence of an algorithm for (s,t)-CVS with run time $f(\omega).n^{O(1)}$, here treewidth is the parameter of interest. Algorithms with running time of this nature are well-studied in the literature and they are called fixed-parameter tractable algorithms in the theory of parameterized complexity [4]. Further, an important lower bound for (s,t)-CVS is the (s,t)-vertex connectivity itself. It is now natural to consider the following parameterization: the size of a (s,t)-CVS minus the (s,t)-vertex connectivity. This type of parameterization is known as above guarantee parameters [48]. We show that (s,t)-CVS parameterized above the (s,t)-vertex connectivity is unlikely to be fixed-parameter tractable and in the terminology of parameterized hardness theory, it is hard for the complexity class $W[2]$ in the W-hierarchy.

5.3 Combinatorial Observations on (s,t)-CVS

We first present the equivalence between a set of edges whose contraction reduces the (s,t)-vertex connectivity to one and a (s,t)-CVS. This gives rise to the question of existence of (s,t)-CVSs. We highlight that there are graphs with no (s,t)-CVSs and we introduce the notion of *inherent* (s,t)-vertex connectivity. For a graph G and $E' \subset E(G)$, let $G \cdot E'$ denote the graph obtained from G by contracting the set E'. To keep $G \cdot E'$ a simple graph multiple edges are removed.

Lemma 5.3.1 *G has $r (r > 0)$ connected components if and only if $G \cdot E'$ has r connected components.*

Proof Observe that any edge contraction does not disconnect a pair of vertices which are already connected. This implies that any edge contraction reduces the size of a connected component, and cannot reduce the number of connected components. Hence the lemma follows. □

Lemma 5.3.2 *G has a (s,t)-CVS S if and only if there exists a set $E' \subset E(G)$ such that the (s,t)-vertex connectivity in $G \cdot E'$ is one. Further, if S is a minimum (s,t)-CVS, then the minimum set of edges to contract is given by a spanning tree of S.*

Proof Consider the graph $G[S]$ and let T_S be a spanning tree of $G[S]$. We know from Lemma 5.1.2, that each edge in any tree is contractible. In particular, each edge e in T_S is contractible. Also, each edge contraction reduces the number of vertices by one. What follows is that for all $e \in E(T_S)$, a sequence of $|V(T_S)| - 1$ edge contractions reduces the (s,t)-vertex connectivity to one. Conversely, let E' denote the set of edges whose contraction reduces the (s,t)-vertex connectivity to one and S' denote the vertex set of E'. Suppose $G[S']$ is not connected and $G[S']$ has at least two connected components. From Lemma 5.3.1, it is observed that any sequence of edge contractions on the set E' leaves a graph in which $G[S']$ has at least two connected components. A contradiction. Therefore, $G[S']$ is connected. Suppose S' is not a (s,t)-vertex separator, then $G \setminus S'$ is connected. This implies that there exists a path P_{st} between s and t in $G \setminus S'$ not containing any element of S'. It is now clear that the (s,t)-vertex connectivity in $G \cdot E'$ is two. A contradiction to the hypothesis that on contracting E', the (s,t)-connectivity becomes one. Therefore, S' is a (s,t)-vertex separator. In particular, S' is a connected (s,t)-vertex separator in G. Hence the lemma. □

The above lemma shows that the minimum number of edges to contract to reduce the (s,t)-vertex connectivity to one is obtained from a spanning tree of a minimum (s,t)-CVS. This result holds good only when such a (s,t)-CVS exists. In the cases when a (s,t)-CVS does not exist, we define the notion of *inherent* (s,t)-vertex connectivity.

5.3.1 Inherent (s,t)-vertex Connectivity: Graphs with no (s,t)-CVS

Not all graphs in general have a (s,t)-CVS. For example, cycle is one such graph where for all (s,t), every (s,t)-vertex separator is an independent set. In the cases when a (s,t)-CVS does not exist, every (s,t)-vertex separator has at least two connected components. In such cases, we define the notion of *inherent* (s,t)-vertex connectivity. Given a graph G with (s,t)-vertex connectivity k, the inherent (s,t)-vertex connectivity is defined to be the smallest $r \leq k$ such that any sequence of edge contractions will result

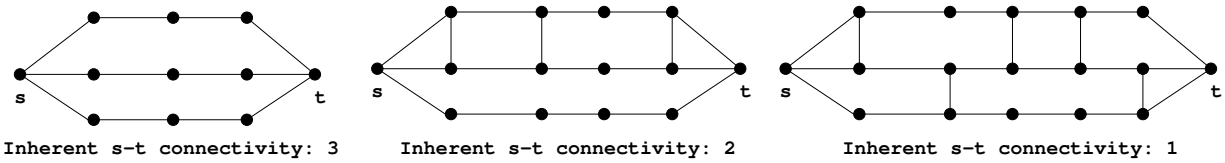


Figure 5.1: An Illustration: Inherent (s,t)-vertex connectivity

in a graph in which the (s,t)-vertex connectivity is r. Further, there is a (s,t)-vertex separator with r connected components and every other (s,t)-vertex separator has at least r connected components. For example, if there exists a connected (s,t)-vertex separator then the inherent (s,t)-vertex connectivity is one and on the other hand if all (s,t)-vertex separators are independent sets of size p then the inherent (s,t)-vertex connectivity is p. Figure 5.1 illustrates three graphs with its inherent (s,t)-vertex connectivity.

We next consider the complexity of finding a minimum (s,t)-CVS when the inherent (s,t)-vertex connectivity is one, and show that it is NP-complete. When the inherent (s,t)-vertex connectivity is more than one too we then need to analyze the complexity of finding a (s,t)-vertex separator with least number of connected components. Interestingly, the decision version of this problem is also NP-complete as minimum (s,t)-CVS is a special case of this problem. Owing to the combinatorial equivalence between the number of edges to be contracted to reduce the (s,t)-vertex connectivity to one and the (s,t)-CVS, all hardness results on (s,t)-CVS carry over to our problem too.

5.4 Complexity of (s,t)-CVS: Hardness and Approximation

We now establish a simple polynomial-time reduction from Steiner tree problem to (s,t)-CVS. Steiner tree problem is shown to be NP-complete in [28]. The decision versions of the Steiner tree problem and the (s,t)-CVS are defined as follows:

Steiner Tree

Given: Graph G, a subset $R \subseteq V(G)$, called terminal set and an integer p
Question: Is there a Steiner tree T with at most p edges containing all of R?

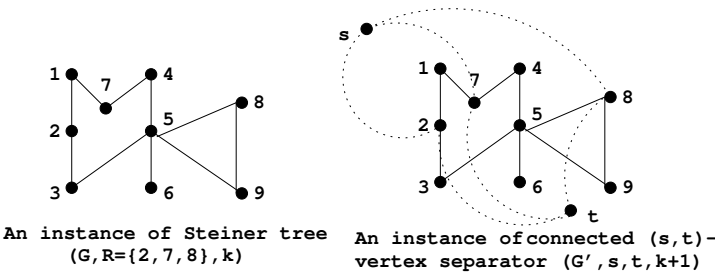


Figure 5.2: Reducing an instance of Steiner tree to an instance of (s,t)-CVS

(s,t)**-CVS**

Given: Graph G, a non-adjacent pair $s,t \in V(G)$, and an integer p
Question: Does there exist a connected (s,t)-vertex separator of size at most p

5.4.1 (s,t)-CVS is NP-complete

Theorem 5.4.1 *Connected (s,t)-vertex separator is NP-complete.*

Proof **(s,t)-CVS is in NP:** Given a certificate $\mathcal{C} = (G,s,t,p)$ where S is a connected (s,t)-vertex separator, to witness the fact that it is in NP, we now present a deterministic polynomial-time algorithm to verify the validity of $\mathcal{C}$. Perform Depth First Search (DFS) on $G[S]$ to check whether $G[S]$ is connected and of size at most p. To verify whether s and t are in two different components in $G \setminus S$ we make use of the fact that both s and t are in the same connected component if and only if there exists a path between s and t. Using this observation, we again run the DFS on $G \setminus S$ from s to check whether there exists a path between s and t. It is known that the DFS runs in time polynomial in the input size [31]. Clearly, our algorithm is a deterministic algorithm which when given a certificate verifies whether it is valid or not, in time polynomial in the input size. Therefore, we conclude that (s,t)-CVS is in NP.

(s,t)-CVS is NP-hard: We present a polynomial-time reduction from Steiner tree problem. Given an instance (G,R,p) of Steiner tree problem, we construct an instance (G',s,t,p') of (s,t)-CVS as follows: $V(G') = V(G) \cup \{s,t\}$, $E(G') = E(G) \cup \{\{s,v\} \mid v \in R\} \cup \{\{t,v\} \mid v \in R\}$, and $p' = p+1$. An example is illustrated in Figure 5.2. We now show that (G,R,p) has a Steiner tree with at most p edges if and only if $(G',s,t,p'=p+1)$ has a (s,t)-CVS of size at most $p+1$. For *only if* claim,

G has a Steiner tree T containing all vertices of R and at most p edges. By our construction of G', to disconnect s and t, we must remove the set $N_{G'}(s)$ which is R, as there is an edge from each element of $N_{G'}(s)$ to t. Since G has a Steiner tree T with at most p edges, implies that T has at most $p+1$ vertices. Clearly, in G', T guarantees a (s,t)-CVS of size at most $p+1$. For *if* claim, G' has a (s,t)-CVS S with at most $p+1$ vertices. Note that any spanning tree on at most $p+1$ vertices has at most p edges. From our construction of G', it follows that $N_{G'}(s) \subseteq S$ and the (s,t)-vertex connectivity is $|N_{G'}(s)|$. This implies that G has a Steiner tree with at most p edges containing $R = N_{G'}(s)$ as the terminal set. Hence the claim. $|V(G')| = |V(G)|+2$ and $|E(G')| \leq |E(G)| + 2|V(G)|$ and the construction of G' takes $O(|E(G)|)$. Hence, this is a polynomial-time reduction. Therefore, we conclude that deciding whether a graph has a (s,t)-CVS is NP-hard. Hence the theorem. □

Since (s,t)-CVS in general graphs is NP-complete, it is natural to analyze the computational complexity of (s,t)-CVS in special graph classes. In the next section, we explore this thought on graphs of chordality c.

5.4.2 (s,t)-CVS on Chordality 5 Graphs is NP-complete

Although (s,t)-CVS in general graphs is NP-complete, (s,t)-CVS is polynomial-time solvable on chordal graphs. This is true because in chordal graphs, every minimal vertex separator is a clique [2]. Now, this line of thought leaves open the complexity of (s,t)-CVS in chordality c graphs. A graph is said to have chordality c, if there is no induced cycle of length at least $c+1$. Note that chordal graphs have chordality 3. We now show that (s,t)-CVS on chordality 5 graphs is NP-complete.

Theorem 5.4.2 *(s,t)-CVS is NP-complete on chordality 5 graphs.*

Proof It is known from [49] that Steiner tree problem on split graphs is NP-complete and this can be reduced in polynomial time to (s,t)-CVS in chordality 5 graphs using the following construction. Note that any split graph G can be seen as a graph with $V(G) = V_1 \cup V_2$ such that $G[V_1]$ is a clique and $G[V_2]$ is an independent set. Also,

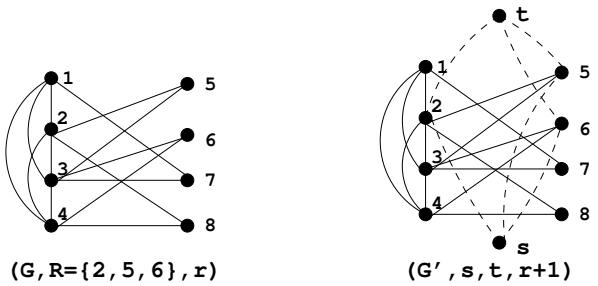


Figure 5.3: Steiner tree in Split graphs maps to (s,t)-CVS in Chordality 5 graphs

split graphs are a subclass of chordal graphs and hence have chordality 3. An instance (G, R, r) of Steiner tree problem on split graphs is reduced to an instance $(G', s, t, r+1)$ of (s,t)-CVS as follows: $V(G') = V(G) \cup \{s,t\}$ and $E(G') = E(G) \cup \{\{s,v\} \mid v \in R\} \cup \{\{t,v\} \mid v \in R\}$. An example is shown in Figure 5.3. We now show that instances created by this transformation have chordality 5. We know that in G any induced cycle is of length at most 3. In G', suppose there exists an induced cycle C of length at least 4. Clearly, C must contain s or t. Let $\{s, u_1, \ldots, u_p\}, p \geq 3$ denote the ordering of vertices in C.

Case 1: $\{u_1, u_p\} \subseteq V_2$. Since the split graph G is connected, there exists $z \in V_1$ such that $\{u_1, z\} \in E(G)$ and there exists $w \in V_1$ such that $\{u_p, w\} \in E(G)$. Note that z and w may denote the same vertex. It is now easy to see that $\{s, u_1, z, w, u_p\}$ induces a cycle of length 4 or 5 in G'. Hence, the chordality of G' is 5.

Case 2: $u_1 \in V_2$ and $u_p \in V_1$. Clearly, there exists z in V_1 such that $\{u_1, z\} \in E(G)$. Now, $\{u_1, z, u_p, s\}$ induces a 4 cycle in G'. This completes our case analysis. Therefore, we conclude that the chordality of G' is 5. Similar to the previous theorem, one can argue that G has a Steiner tree with at most r edges if and only if G' has a (s,t)-CVS with at most $r+1$ vertices. As a consequence, we see that (s,t)-CVS on chordality 5 graphs is NP-hard. Moreover, an argument similar to the previous theorem shows that it is in NP. Thus, we conclude (s,t)-CVS in chordality 5 graphs is NP-complete. □

5.4.3 Hardness of Approximation for (s,t)-CVS

Our goal is to show that there is no polynomial-time approximation algorithm for (s,t)-CVS with approximation factor $\delta.log^{2-\epsilon}n$ for some $\delta > 0$ and for any $\epsilon > 0$, unless NP

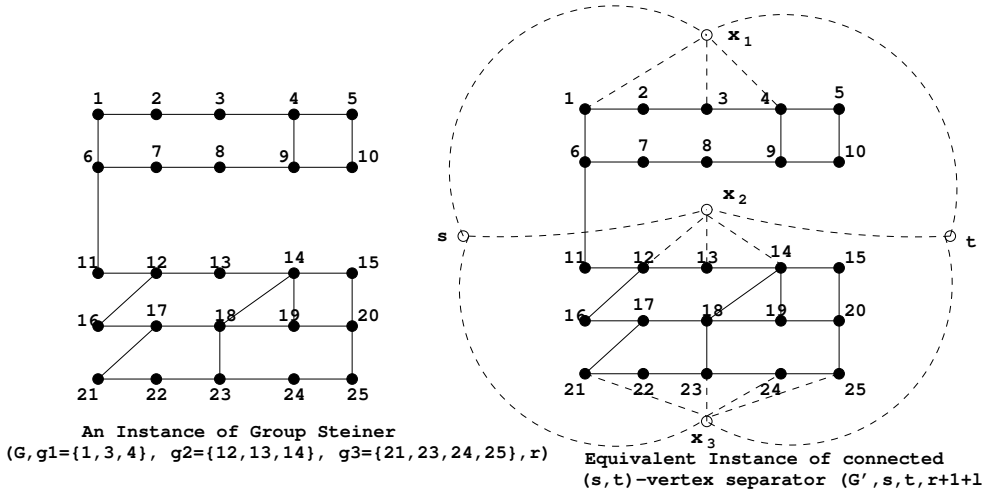


Figure 5.4: An instance of Group Steiner tree reduces to an instance of (s,t)-CVS

has quasi-polynomial Las-Vegas algorithms. i.e. (s,t)-CVS is $\Omega(log^{2-\epsilon}n)$-hard for any $\epsilon > 0$.

The Group Steiner tree problem can be stated as follows: given a graph G and a subsets of vertices, which we call groups $g_1, g_2, \ldots, g_l \subseteq V(G)$, the objective is to find a minimum cost subtree T of G that contains at least one vertex from each group g_i. The Group Steiner tree problem is a generalization of the Steiner tree problem and therefore, it is NP-complete [28]. Further, the group Steiner tree problem with l groups is at least as hard as the set cover problem, and thus can not be approximated to a factor $o(\log l)$, unless $P = NP$ [50]. On the hardness of approximation due to [51], we have the following result: Group Steiner tree problem is $\Omega(\log^{2-\epsilon} n)$-hard, for all $\epsilon > 0$ unless NP has quasi-polynomial Las-Vegas algorithms.

Given an instance of group Steiner tree $(G, g_1, g_2, \ldots, g_l \subseteq V(G), r)$, we construct an instance $(G', s, t, l+r+1)$ of (s,t)-CVS as follows: $V(G') = V(G) \cup \{s,t\} \cup \{x_i \mid 1 \leq i \leq l\}$. $E(G') = E(G) \cup \{\{s, x_i\} \mid 1 \leq i \leq l\} \cup \{\{t, x_i\} \mid 1 \leq i \leq l\} \cup \{\{x_i, y\} \mid y \in g_i \text{ and } 1 \leq i \leq l\}$. An example is illustrated in Figure 5.4.

Theorem 5.4.3 *Let G be an instance of Group Steiner tree on l groups and G' be an instance of (s,t)-CVS. G has a Group Steiner tree with at most r edges if and only if G' has a (s,t)-CVS of size at most $r+1+l$.*

Proof We first prove the necessity. Given that G has a Group Steiner tree T with at most r edges that contains at least one vertex from each group g_i. By the construction

of G', it is clear that the (s,t)-vertex connectivity is l. This implies that any (s,t)-CVS in G' must have at least l vertices. Therefore, the l new vertices together with $r+1$ vertices in T form a (s,t)-CVS of size at most $r+1+l$ in G'. Conversely, by the construction of G', any (s,t)-vertex separator must contain all x_i's. In particular, any (s,t)-CVS S must contain at least l vertices. This is true because of two reasons. One by the construction of G', there are no edges between any pair of x_i and x_j. Also, S must contain at least one element of $N_{G'}(x_i)$ for each x_i. Given that G' has a (s,t)-CVS S of size at most $r+1+l$, it follows that S has at most $r+1$ vertices of G and any spanning tree on such $r+1$ vertices is a Group Steiner tree with at most r edges. Hence the theorem. □

Remark: The above reduction is an approximation ratio preserving reduction. Let OPT_g and OPT_c denote the size of any optimum solution of Group Steiner tree problem and (s,t)-CVS problem, respectively. Note that $OPT_c = OPT_g + l$ and $OPT_g \geq l$. Suppose there is an $(1+\alpha)$-approximation algorithm for (s,t)-CVS, where $\alpha \leq \delta.log^{2-\epsilon}n$, $\delta, \epsilon > 0$. Then the output of the algorithm is $(1+\alpha)OPT_c = (1+\alpha)(OPT_g + l) \leq (1+\alpha)(OPT_g + OPT_g) = 2(1+\alpha)OPT_g$. This implies, a $2(1+\alpha)$-approximation algorithm for the Group Steiner tree problem, which is unlikely, unless NP has quasi-polynomial Las-Vegas algorithms [51].

5.4.4 $(\lceil \frac{c}{2} \rceil)$-Approximation on Chordality c Graphs

The focus of this section is to present a polynomial-time approximation algorithm with ratio $(\lceil \frac{c}{2} \rceil)$ for (s,t)-CVS on graphs with chordality c. Prior to this, we present an observation on the structure of minimal vertex separators in chordality c graphs.

Lemma 5.4.4 *Let G be a graph of chordality c. For each minimal vertex separator S, for each $u, v \in S$ such that $\{u,v\} \notin E(G)$, there exists a path of length at most $\lceil \frac{c}{2} \rceil$ whose internal vertices are in C_s or C_t, where C_s and C_t are components in $G \setminus S$ containing s and t, respectively.*

Proof Suppose for some non-adjacent pair $\{u,v\} \subseteq S$, both P^1_{uv} and P^2_{uv} are of length more than $\lceil \frac{c}{2} \rceil$, where P^1_{uv} and P^2_{uv} are shortest paths from u to v whose internal vertices

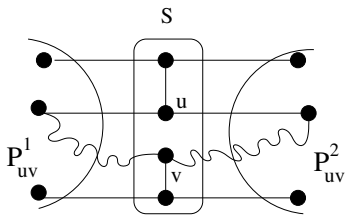


Figure 5.5: An Illustration for the proof of Lemma 5.4.4

are in C_s and C_t, respectively. An illustration is shown in Figure 5.5. Now, there is an induced cycle C containing u and v such that $|C| > \lceil \frac{c}{2} \rceil + \lceil \frac{c}{2} \rceil \geq c$. However, this contradicts the fact that G is of chordality c. Hence the lemma. □

Let OPT denote the size of any minimum (s,t)-CVS on chordality c graphs. Clearly, $OPT \geq k$, where k is the (s,t)-vertex connectivity. The description of approximation algorithm ALG is as follows:

1. Compute a minimum (s,t)-vertex separator S in G. Let $S = (v_1, \ldots, v_k)$ be an arbitrary ordering of vertices in S.

2. For each non-adjacent pair $\{v_i, v_{i+1}\} \subseteq S, 1 \leq i \leq k-1$, find a path $P_{v_i v_{i+1}}$ of length at most $\lceil \frac{c}{2} \rceil$ whose internal vertices are in C_s or C_t. Such a path exists as per Lemma 5.4.4. $S' = \bigcup_{1 \leq i \leq k-1} V(P_{v_i v_{i+1}}) \cup S$.

Observe that S' is a (s,t)-CVS in G. The upper bound on the size of S' output by ALG is: $|S'| \leq k + (k-1)(\lceil \frac{c}{2} \rceil - 1)$. Therefore, approximation ratio β is

$$\beta \leq \frac{k+(k-1)(\lceil \frac{c}{2} \rceil - 1)}{k} = 1 + (1 - \frac{1}{k})(\lceil \frac{c}{2} \rceil - 1) < 1 + (\lceil \frac{c}{2} \rceil - 1) = \lceil \frac{c}{2} \rceil$$

5.5 A Polynomial-time Algorithm for (s,t)-CVS in Bipartite Chordality 4 Graphs

We have already shown that (s,t)-CVS in chordality 3 graphs is polynomial-time solvable, whereas it is NP-complete on chordality 5 graphs. This observation leaves open the complexity status of (s,t)-CVS in chordality 4 graphs. We show that on bipartite chordality 4 graphs, (s,t)-CVS is polynomial-time solvable. Note that bipartite

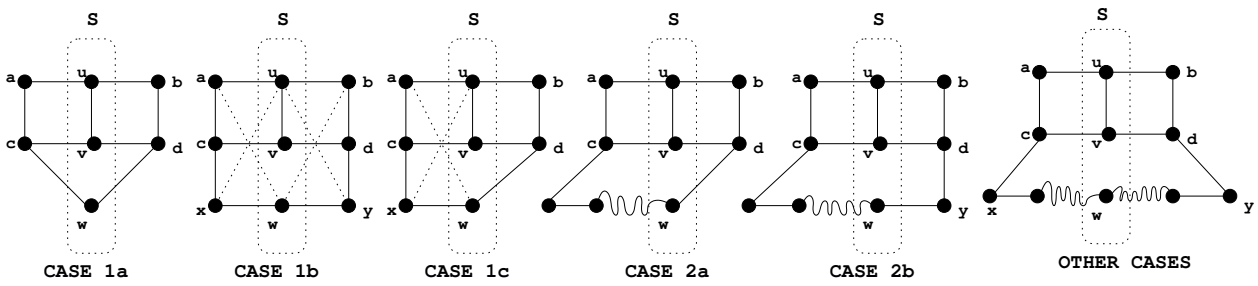


Figure 5.6: An illustration for the proof of Theorem 5.5.1

chordality 4 graphs are popularly known as chordal bipartite graphs in the literature [2]. A bipartite graph G is chordal bipartite, if there is no induced cycle of length at least 6 in G. We first show that each minimal (s,t)-vertex separator in a chordal bipartite graph is either connected or an independent set. Using which we establish the result that (s,t)-CVS in chordal bipartite graphs is polynomial-time solvable. Let G be a chordal bipartite graph. Recall that for a minimal (s,t)-vertex separator S, C_s and C_t denote the components in $G \setminus S$ containing s and t, respectively. $\tilde{P}_{xy}$ denotes a path on the vertex set $V(P_{xy}) \setminus \{x\}$.

Theorem 5.5.1 *Let G be a chordal bipartite graph. Every minimal (s,t)-vertex separator S in G is such that $G[S]$ is either an independent set or a connected subgraph.*

Proof If $|S| \leq 2$, then our claim follows. For $|S| \geq 3$, we present a proof by contradiction. Suppose there exists a minimal (s,t)-vertex separator S such that $G[S]$ is neither an independent set nor a connected subgraph. We must find $u, v, w \in S$ such that $\{u,v\} \in E(G)$ and $\{u,w\} \notin E(G), \{v,w\} \notin E(G)$. Since S is a minimal (s,t)-vertex separator, there exists $a \in V(C_s) \cap N_G(u)$ and there exists $b \in V(C_t) \cap N_G(u)$. Similarly, $c \in V(C_s) \cap N_G(v)$, $d \in V(C_t) \cap N_G(v)$, $x \in V(C_s) \cap N_G(w)$, and $y \in V(C_t) \cap N_G(w)$. Since G is chordal bipartite and C_s is a connected component, it follows that $\{a,c\} \in E(G)$ and $\{b,d\} \in E(G)$. Consider a shortest path P_{cw} between c and w and a shortest path P_{dw} between d and w. To complete the proof, we present a case analysis by considering the parity of P_{cw} and P_{dw}. An illustration is given in Figure 5.6.

Case 1: Both $|P_{cw}| \leq 3$ and $|P_{dw}| \leq 3$.

Case 1a: $|P_{cw}| = 2$ and $|P_{dw}| = 2$. i.e. $\{c,w\}, \{d,w\} \in E(G)$. Since G is bipartite,

clearly $\{a, w\}, \{b, w\} \notin E(G)$. Now the set $\{u, a, c, w, d, b\}$ induces a cycle of length 6 in G.

Case 1b: $|P_{cw}| = 3$ and $|P_{dw}| = 3$. Assume $V(P_{cw}) = \{c, x, w\}$ and $V(P_{dw}) = \{d, y, w\}$. Clearly, the set $\{v, c, x, w, y, d\}$ induces a cycle of length 6. Note that this observation is true irrespective of whether the edges $\{a, w\}, \{b, w\}, \{u, x\}$, and $\{u, y\}$ are contained in $E(G)$ or not.

Case 1c: $|P_{cw}| = 3$ and $|P_{dw}| = 2$. Here, we see that the set $\{v, c, x, w, d\}$ induces a cycle of length 5. The argument for the case $|P_{cw}| = 2$ and $|P_{dw}| = 3$ is symmetric.

Case 2: Exactly one of P_{cw} or P_{dw} is of size at most 3 and the other is of size at least 4, say, $|P_{cw}| \geq 4$ and $|P_{dw}| \leq 3$.

Case 2a: $|P_{dw}| = 2$. If both u and a are not adjacent to any vertex in P_{cw}, then the set $\{u, a, b, d\} \cup V(P_{cw})$ induces a cycle of length at least 8. If $\{u, z\} \in E(G)$ for some z in P_{cw}, then choose a z such that there is no z' in $\tilde{P}_{zw}$, $\{u, z'\} \in E(G)$. Clearly, $\{u, b, d\} \cup V(P_{zw})$ induces a cycle of length at least 5. Otherwise, $\{u, z\} \notin E(G)$ for any z in P_{cw} and a is adjacent to some vertex z in $\tilde{P}_{cw}$. In this case, identify a z such that there is no z' in $\tilde{P}_{zw}$, $\{a, z'\} \in E(G)$. Now, the set $\{a, u, b, d\} \cup V(P_{zw})$ induces a cycle of length at least 6.

Case 2b: $|P_{dw}| = 3$. Let z be a vertex in P_{cw} such that $\{v, z\} \in E(G)$ and there is no z' in $\tilde{P}_{zw}$, $\{a, z'\} \in E(G)$. It is easy to see that $V(P_{dw}) \cup \{v\} \cup V(P_{zw})$ induces at least a 5 cycle. A contradiction in this case too.

We next consider the cases in which $|P_{cw}| \geq 4$ and $|P_{dw}| \geq 4$. i.e., $|P_{xw}| \geq 3$ and $|P_{yw}| \geq 3$. Let $V(P_{xw}) = \{x = x_1, \ldots, x_p = w\}, p \geq 3$ and $V(P_{yw}) = \{y = y_1, \ldots, y_q = w\}, q \geq 3$.

Case 3: Both $|P_{xw}|$ and $|P_{yw}|$ are odd. If $X = \{v\} \cup V(P_{cw}) \cup V(P_{dw})$ does not induce a chordal bipartite subgraph, then we easily arrive at a contradiction. Otherwise, we assume X induces a chordal bipartite subgraph. Now, we must analyze the edges from $\{u, a\}$ to $V(P_{xw})$ and from $\{u, b\}$ to $V(P_{yw})$. Since the graph induced on $\{v\} \cup V(P_{cw}) \cup V(P_{dw})$ is chordal bipartite, the only possible chords from v to $V(P_{cw})$ and $V(P_{dw})$ are the following: $\{\{v, x_i\} \mid i \bmod 2 = 0\}$ and $\{\{v, y_i\} \mid i \bmod 2 = 0\}$.

Case 3a: there exists x_i in P_{xw} and y_j in P_{yw} such that $\{u, x_i\} \in E(G)$ and $\{u, y_j\} \in E(G)$. Consider the vertex x_i such that there is no $x_k, i < k < p$ and $\{x_k, u\} \in E(G)$. Similarly, y_j be the vertex such that there is no $y_k, j < k < q$ and $\{y_k, u\} \in E(G)$. Since $\{v, x_{p-1}\} \in E(G)$, it implies that $\{u, x_{p-1}\} \notin E(G)$. Similarly, $\{u, y_{q-1}\} \notin E(G)$. It is now clear that $\{u\} \cup V(P_{x_iw}) \cup V(P_{y_jw})$ induces a cycle of length at least 6 in G.

Case 3b: for any x_i in P_{xw} and for any y_j in P_{yw} neither $\{u, x_i\} \in E(G)$ nor $\{u, y_j\} \in E(G)$. In this case, we analyze the edges from a to P_{xw}. If $\{a, w\} \in E(G)$ then $V(P_{xw}) \cup \{a, c\}$ is a cycle of odd length, contradicting the fact that G is bipartite. Therefore, $\{a, w\} \notin E(G)$. Let x_i be the vertex in P_{xw} such that $\{a, x_i\} \in E(G)$ and there is no $x_k \in V(P_{xw}), i < k < p$ such that $\{a, x_k\} \in E(G)$. If for any y_j, $\{b, y_j\} \notin E(G)$ then $\{a, x_i, u, b\} \cup V(P_{dw})$ induces a cycle of length at least 8 in G. Otherwise, y_j be the vertex in P_{yw} such that $\{b, y_j\} \in E(G)$ and there is no $y_k \in V(P_{yw}), j < k < q$ such that $\{b, y_k\} \in E(G)$. Then, $\{a, u, b\} \cup V(P_{y_j w}) \cup V(P_{x_i w})$ induces a cycle of length at least 6 in G.

Case 3c: there exists x_i in P_{xw} such that $\{u, x_i\} \in E(G)$. In this case, similar to case 3b, by analyzing the edges from b to P_{yw} we can construct an induced cycle of length at least 5. Again a contradiction.

Case 4: Both $|P_{xw}|$ and $|P_{yw}|$ are even. In this case, we show that $\{v\} \cup V(P_{cw}) \cup V(P_{dw})$ does not induce a chordal bipartite subgraph irrespective of edges from $\{a, u, b\}$ to P_{xw} and P_{yw}. If $\{v, d, y\}$ induces a triangle, then we are done. Otherwise, let y_j be the vertex in P_{yw} such that $\{v, y_j\} \in E(G)$ and there is no y_k, $j < k < q$ such that $\{v, y_k\} \in E(G)$. Similarly, there exists x_i in P_{xw} such that $\{v, x_i\} \in E(G)$ and there is no x_k, $i < k < p$ such that $\{v, x_k\} \in E(G)$. The set $\{v\} \cup V(P_{x_i w}) \cup V(P_{y_j w})$ induces a cycle of length at least 5. In the last configuration, suppose we find edges $\{v, x_i\}, \{v, x_j\}$ such that $|i - j| \geq 3$ and there are no edges $\{v, x_k\}, i < k < j$, then clearly we see that there exists an induced cycle of length at least 5 in G. If $|P_{xw}|$ is even and $|P_{yw}|$ is odd, then $V(P_{xw}) \cup V(P_{yw}) \cup \{c, v, d\}$ is a cycle of odd length, contradicting the fact that G is bipartite.

Since we arrive at a contradiction in all cases, it follows that our assumption is wrong. Therefore, every minimal vertex separator S in G is such that $G[S]$ is either an independent set or a connected subgraph. Hence the theorem. □

Theorem 5.5.2 *Let G be a chordal bipartite graph and S be a minimal vertex separator S in G. If $G[S]$ is an independent set, then in $G \setminus S$, there exists u in C_s and there exists v in C_t such that $S \subseteq N_G(u)$ and $S \subseteq N_G(v)$.*

Proof Our proof is by induction on $n = |V(G)|$.

Base case: Graphs in which for all S, $|S| \leq 2$. If $|S| = 1$, then S contains a cut vertex and our claim is true. If $|S| = 2$, then either $G[S]$ is connected or as per Lemma 5.4.4, we find u in C_s such that $S \subseteq N_G(u)$ and v in C_t such that $S \subseteq N_G(v)$.

Hypothesis: Assume all chordal bipartite graphs on $n-1$ vertices satisfy our claim.

Induction Step: Consider a chordal bipartite graph G on n vertices. Let S be a minimal (s,t)-vertex separator in G such that $|S| \geq 3$. Let $S = \{x, y, u_1, \ldots, u_p\}, p \geq 1$. Consider the graph $G \cdot xy$ obtained by contracting the non-adjacent pair $\{x, y\}$. $S' = S \setminus \{x, y\} \cup \{z_{xy}\}$ and edges incident on x or y are now incident on z_{xy}. Observe that S' is a minimal (s,t)-vertex separator in $G \cdot xy$. Clearly, $|V(G \cdot xy)| = |V(G)| - 1$ and hence, by the induction hypothesis, in $G \cdot xy$, there exists u in C'_s and v in C'_t satisfying our claim, where C'_s and C'_t are connected components in $(G \cdot xy) \setminus S'$ containing s and t, respectively. We now prove in G the existence of vertex u in C_s satisfying our claim. The proof for the existence of vertex v in C_t is symmetric. If $\{u, x\}, \{u, y\} \in E(G)$, then clearly u and v are desired vertices in G as well. Otherwise, assume $y \in N_G(u)$ and $x \notin N_G(u)$. Note that $S \setminus \{x\} \subset N_G(u)$. Let P^s_{xu} denote a shortest path between x and u such that the internal vertices are in C_s. Consider the vertex w in P^s_{xu} such that $\{x, w\} \in E(G)$. Such a w exists as S is a minimal (s,t)-vertex separator in G. If for all $z \in S$, $\{w, z\} \in E(G)$, then w is a desired vertex in C_s. Otherwise, there exists $z \in S$ such that $\{w, z\} \notin E(G)$. Let P^s_{wu} denote a subpath of P^s_{xu} on the vertex set $\{w = w_1, \ldots, w_q = u\}, q \geq 2$. If for any $2 \leq i \leq q-1$, $\{z, w_i\} \notin E(G)$, then $P^s_{xu}\{u, z\}P^t_{xz}$ form an induced cycle of length at least 5 in G where P^t_{xz} denote a shortest path between x and z such that the internal vertices are in C_t. Otherwise, for some $2 \leq i \leq q-1$, $\{z, w_i\} \in E(G)$. Let i be the smallest integer such that $\{z, w_i\} \in E(G)$. In this case, $P^s_{xw_i}\{w_i, z\}P^t_{xz}$ form an induced cycle of length at least 5 in G, where $P^s_{xw_i}$ denote a subpath of P^s_{xu} on the vertex set $\{x, w = w_1, \ldots, w_i\}, 2 \leq i \leq q-1$. However, we know that G is chordal bipartite. A contradiction. Therefore, there exists u in C_s such that $S \subseteq N_G(u)$. Similarly, we can argue the existence of a vertex v in C_t. This completes our case analysis and the induction is complete. Hence the theorem. □

Lemma 5.5.3 *The cardinality of any minimum (s,t)-CVS in a chordal bipartite graph with the (s,t)-vertex connectivity k is either k or $k+1$.*

Proof Note that any minimum (s,t)-CVS is of size at least k as the (s,t)-vertex connectivity is k. From Theorem 5.5.1, we know that any minimum (s,t)-vertex separator S is either connected or an independent set. This implies that, if there exists S such that $G[S]$ is connected then there exists a minimum (s,t)-CVS of size k. Otherwise, every

S is such that $G[S]$ is an independent set. In this case, we know from Theorem 5.5.2 that there exists u in C_s such that $S \subseteq N_G(u)$. It is clear that $S \cup \{u\}$ is a minimum (s,t)-CVS of size $k+1$. Hence the claim. □

Using the fact that chordal bipartite graphs have polynomial number of minimal vertex separators [52], it follows from Theorems 5.5.1 and 5.5.2 that (s,t)-CVS is polynomial-time solvable.

5.6 Parameterized Complexity of (s,t)-CVS

From the parameterized setting, we report two results by considering the following parameterizations: parameterizing above the (s,t)-vertex connectivity and treewidth of the graph. In Section 5.6.1, we show that (s,t)-CVS parameterized above the (s,t)-vertex connectivity is hard for the complexity class $W[2]$ and in Section 5.6.2, we show that (s,t)-CVS parameterized by treewidth is FPT.

5.6.1 (s,t)-CVS Parameterized above the (s,t)-vertex connectivity is $W[2]$-hard

We consider the following parameterization which is the size of (s,t)-CVS minus the (s,t)-vertex connectivity. Since the size of every (s,t)-CVS is at least the (s,t)-vertex connectivity, it is natural to parameterize above the (s,t)-vertex connectivity and its parameterized version is defined below.

(s,t)-CVS Parameterized above the (s,t)-vertex connectivity

Instance: A graph G, a pair $\{s,t\} \notin E(G)$ with (s,t)-vertex connectivity k and $r \in \mathbb{Z}^+$
Parameter: r
Question: Is there a (s,t)-vertex separator $S \subset V(G)$, $|S| \leq k+r$ such that $G[S]$ is connected?

This type of parameterization is known as parameterizing above guaranteed values and it was introduced by Mahajan and Raman in [48]. For (s,t)-CVS, the guaranteed value,

i.e. the lower bound for any (s,t)-CVS is the (s,t)-vertex connectivity itself.

We now show that (s,t)-CVS is unlikely to be fixed-parameter tractable when parameterized above the (s,t)-vertex connectivity under standard parameterized complexity assumption [3]. In particular, we show that it is $W[2]$-hard. We now present a parameterized reduction from parameterized Steiner tree problem to (s,t)-CVS parameterized above the (s,t)-vertex connectivity. This parameterized version of Steiner tree problem is shown to be $W[2]$-hard in [3] and it is proved to be in $W[2]$ in [53]. We also highlight the fact that if $|R|$ is a parameter then Steiner tree problem belongs to FPT [54] by a dynamic programming approach running in time $O(3^{|R|}n^{O(1)})$.

Parameterized Steiner tree problem

Instance: A graph G, a terminal set $R \subseteq V(G)$, and an integer p

Parameter: p

Question: Is there a set of vertices $T \subseteq V(G) \setminus R$ such that $|T| \leq p$ and $G[R \cup T]$ is connected? T is called the Steiner set (Steiner vertices).

Theorem 5.6.1 *(s,t)-CVS Parameterized above the (s,t)-vertex connectivity is $W[2]$-hard.*

Proof Given an instance (G,R,r) of Steiner tree problem, we construct an instance (G',s,t,k,r) of (s,t)-CVS with the (s,t)-vertex connectivity $k = |R|$ as follows: $V(G') = V(G) \cup \{s,t\}$ and $E(G') = E(G) \cup \{\{s,v\} \mid v \in R\} \cup \{\{t,v\} \mid v \in R\}$. We now show that (G,R,r) has a Steiner tree with at most r Steiner vertices if and only if (G',s,t,k,r) has a (s,t)-CVS of size at most $k+r$. For *only if* claim, G has a Steiner tree T containing all vertices of R and at most r Steiner vertices. By our construction of G', to disconnect s and t, we must remove the set $N_{G'}(s)$ which is R, as there is an edge from each element of $N_{G'}(s)$ to t. Since G has a Steiner tree with at most r Steiner vertices, it implies that in G', it guarantees a (s,t)-CVS of size at most $k+r$. For *if* claim, G' has a (s,t)-CVS S with at most $k+r$ vertices. Since the (s,t)-vertex connectivity is k and S is a (s,t)-vertex separator, from our construction of G', it follows that $N_{G'}(s) \subseteq S$ and $k = |N_{G'}(s)|$. This implies that G has a Steiner tree with $R = N_{G'}(s)$ as the terminal set and $S \setminus N_{G'}(s)$ as the Steiner vertices of size at most r. Hence the claim. $|V(G')| = |V(G)| + 2$ and $|E(G')| \leq |E(G)| + 2|V(G)|$ and the construction

of G' takes $O(|E(G)|)$. Clearly, the reduction is a parameter preserving parameterized reduction. Therefore, we conclude that deciding whether a graph has a (s,t)-CVS is $W[2]$-hard with parameter r. Hence the theorem. □

5.6.2 (s,t)-CVS Parameterized by treewidth is FPT

We transform an instance of (s,t)-CVS problem to the satisfiability of a formula in *monadic second order logic* (MSOL). It is well-known that by the application of Courcelle's Theorem [55], a problem expressible in MSOL can be solved in linear time on bounded treewidth graphs. Therefore, (s,t)-CVS on bounded treewidth graphs is linear time. A recent paper by Marx [26] uses the same approach to prove the fixed-parameter tractability of other constrained separator problems. We now present the description of MSOL formulation for (s,t)-CVS. The atomic predicates used are as follows: For a set $S \subseteq V(G)$, $S(v)$ denotes that v is an element of S, and the predicate $E(u,v)$ denotes the adjacency between u and v in G. $T = \{s,t\}$ and k is the upper bound on the size of the desired (s,t)-CVS. We construct the formula ϕ in MSOL as

$$\phi = \exists S(AtMost_k(S) \wedge Separates(S) \wedge ConnectedSubgraph(S))$$

Here the predicate $AtMost_k(S)$ is true if and only if $|S| \leq k$, $Separates(S)$ is true if and only if S separates the vertices of T in G, and $ConnectedSubgraph(S)$ is true if and only if S induces a connected subgraph in G. We refer [26] for formulae $AtMost_k(S)$ and $Separates(S)$. The formula $Connects(Z,s,t)$ is due to [56], where $Connects(Z,s,t)$ is true if and only if in G, there is a path from s and t all vertices of which belong to Z. $ConnectedSubgraph(S)$ is true if and only if for every subset S' of S there is an edge between S' and $S \setminus S'$.

- $AtMost_k(S) : \forall c_1, \ldots, \forall c_{k+1} \bigvee_{1 \leq i,j \leq k+1} (c_i = c_j)$
- $Separates(S) : \forall s \forall t \forall Z(T(s) \wedge T(t) \wedge \neg(s=t) \wedge \neg S(s) \wedge \neg S(t) \wedge Connects(Z,s,t)) \rightarrow (\exists v(S(v) \wedge Z(v)))$, $Connects(Z,s,t) : Z(s) \wedge Z(t) \wedge \forall P((P(s) \wedge \neg P(t)) \rightarrow (\exists v \exists w(Z(v) \wedge Z(w) \wedge P(v) \wedge \neg P(w) \wedge E(v,w))))$
- $ConnectedSubgraph(S) : \forall S' \subseteq S((S \neq S') \wedge (\exists u(S(u) \wedge S'(u))) \rightarrow (\exists u \exists w(S(u) \wedge S(w) \wedge (u,w) \in E(G) \wedge S'(v) \wedge \neg S'(w)))$

This completes the observation that in bounded treewidth graphs, a minimum (s,t)-CVS can be found in linear time.

5.7 A Related Problem: Reducing Connectivity by r $(r \geq 1)$ using Edge Contractions

We shall now analyze the complexity of the following computational problem: find a minimum number of edges whose contraction reduces the (s,t)-vertex connectivity by r $(r \geq 1)$. The decision version of the problem is stated as follows:

Instance: A graph G with (s,t)-vertex connectivity k and $l, r \in \mathbb{Z}^+$
Question: Is there a set $E' \subset E(G)$, $\|E'\| \leq l$ such that $\kappa(G \cdot E') = k - r$

Prior to the complexity analysis, we shall present some combinatorial observations relating this problem and (s,t)-CVS.

Observation 5.7.1 *G has a (s,t)-CVS if and only if there exists a (s,t)-vertex separator S such that $G[S]$ has exactly one connected component.*

Lemma 5.7.1 *Let G be a graph with (s,t)-vertex connectivity k. For $r \geq 1$, there exists a (s,t)-vertex separator S in G such that $G[S]$ has at most $k - r$ connected components if and only if there exists $E' \subset E(G)$ whose contraction reduces the (s,t)-vertex connectivity by r (i.e. the (s,t)-vertex connectivity becomes $k - r$).*

Proof *Necessity:* Suppose there does not exist $E' \subset E(G)$ whose contraction reduces the (s,t)-vertex connectivity by r. This implies that for any $E' \subset E(G)$, the (s,t)-vertex connectivity in $G \cdot E'$ is at least $k - r + 1$. This implies that the inherent (s,t)-vertex connectivity is at least $k - r + 1$ and every (s,t)-vertex separator has at least $k - r + 1$ connected components. A contradiction. Conversely, suppose every (s,t)-vertex separator S in G is such that $G[S]$ has at least $k - r + 1$ connected components. This implies that the inherent (s,t)-vertex connectivity is at least $k - r + 1$. Moreover, for any subset E' of edges, the (s,t)-vertex connectivity in $G \cdot E'$ is at least $k - r + 1$, contradicting the hypothesis. This completes the sufficiency. □

In the light of Lemma 5.7.1, the problem of reducing the (s,t)-vertex connectivity by r $(r \geq 1)$ using a minimum number of edge contractions is computationally equivalent to the following:

Instance: A graph G with (s,t)-vertex connectivity k and $r \in \mathbb{Z}^+$
Question: Is there a (s,t)-vertex separator S such that $G[S]$ has at most $k-r$ connected components and S is minimum

Note that in the above problem when $r = k-1$ it is actually the (s,t)-CVS problem. Therefore, the problem of reducing the (s,t)-vertex connectivity by r is indeed NP-complete.

Concluding Remarks

In this chapter, we have investigated the complexity of connected (s,t)-vertex separator ((s,t)-CVS) and showed that for every $\epsilon > 0$, (s,t)-CVS is $\Omega(log^{2-\epsilon}n)$-hard, unless NP has quasi-polynomial Las-Vegas algorithms. Also presented, a polynomial-time algorithm for (s,t)-CVS on chordal bipartite graphs and showed that (s,t)-CVS is NP-complete on graphs with chordality at least 5. Moreover, from a parameterized-view, we established the fact that parameterizing above the (s,t)-vertex connectivity is $W[2]$-hard. In the next chapter, we study the impact of edge additions on vertex connectivity by considering the computational problem of increasing the vertex connectivity by one using a minimum number of edge additions.

CHAPTER 6

A Framework for Connectivity Augmentation

In this chapter, the goal is to introduce a unified framework for connectivity augmentation. We report our study on connectivity augmentation in 1-connected graphs, 2-connected graphs, k-trees, and k-connected chordal graphs. We first represent the graph under consideration using a 'tree-like' graph. This tree is unique and explicitly captures the connectivity information of the graph. Using this tree, our proposed data structure maintains the set of equivalence classes based on an equivalence relation on the set of leaves of the tree. This partition determines a set of edges to be augmented to increase the connectivity of the graph by one. Based on our data structure, we present a new combinatorial analysis and an elegant proof of correctness of our algorithm for optimum connectivity augmentation.

6.1 A Survey on Connectivity Augmentation: Past Results and Motivation

Connectivity augmentation in graphs is a classical topic of combinatorial optimization. Vertex connectivity augmentation of a graph adds the smallest set of edges to reach a given vertex connectivity. A special case of this study is to increase the connectivity by one. i.e., given a k-vertex connected graph G, find a minimum number of edges to be augmented to G to increase its connectivity to $k+1$. This was an open problem for the past three decades, only recently, it was shown by Vegh in [57] that it is polynomial-time solvable. Although, the complexity of this problem was open for several years, special cases of this problem have attracted many researchers.

Biconnectivity Augmentation: Eswaran and Tarjan [58] initiated the study of biconnectivity augmentation which asks for identifying a minimum set of edges whose augmentation makes the given graph 2-vertex connected. The following results are presented in [58] for biconnectivity augmentation. From a connected graph G, a *block tree*

T is constructed in linear time. A block tree is a tree such that each node is one of the following: a cut-vertex, a cut-edge, and a maximal 2-connected subgraph in G. Using T, Eswaran and Tarjan [58] presented the lower bound on the optimum biconnectivity augmentation number of G which is $\max\{\lceil \frac{l}{2} \rceil, \Delta_c(T) - 1\}$, where l is the number of leaves in T and $\Delta_c(T)$ denotes the maximum degree among all cut vertices of G. In [58], a constructive proof of tightness of the lower bound is also mentioned and using which an $O(n + m)$ time algorithm can be designed for biconnectivity augmentation. Rosenthal and Goldner [59] developed an $O(n + m)$ linear-time sequential algorithm for biconnectivity augmentation based on the combinatorial analysis given in [58]. The algorithm given in [59] involves three stages: stage-1 augments a set of edges such that the resulting graph is connected, the graph resulting from stage-2 is such that the degree of each vertex is at most half the number of leaves of the block tree and the graph obtained from stage-3 is biconnected. Later, Hsu and Ramachandran [60] discovered an error in stage-3 of the algorithm reported in [59] and they presented the corrected version in [60]. Subsequently, they proved that the bound mentioned in [58] is tight. In [60], a parallel algorithm for biconnectivity augmentation was also presented. Recently, Hsu in [61], presented a simple sequential algorithm and an improved parallel algorithm for biconnectivity augmentation and these improvements are over the algorithm presented in [60].

A Research Gap: It is important to note that the approach in [60] iteratively determines the edge to be augmented by recomputing the associated tree. As a result, the idea of [60] yields a not so elegant combinatorial analysis and makes the implementation a challenging task. The idea in [61], though avoids recomputation of the associated tree, the underlying combinatorial analysis is quite involved. It may be noted that, despite the efforts of [59, 60, 61], a simple linear-time algorithm without recomputing the associated tree is still open. This is the first motivation behind our work.

Triconnectivity Augmentation: Another fundamental graph theoretic problem in this line is triconnectivity augmentation which determines a minimum number of edges to be augmented to a biconnected graph to make it 3-vertex connected. This study was initiated by Watanabe and Nakamura in [62], where similar to biconnectivity augmentation, using the structural understanding of a biconnected graph, a *3-block tree* is constructed. A 3-block tree is a tree in which each node is one of the following: a maximal

3-connected component, a 2-size vertex separator, and a polygon. A lower bound on the size of the triconnectivity augmentation involving leaves and maximum degree of a 3-block tree is proposed in [62]. This lower bound is similar to the one proposed in [58] for biconnectivity augmentation. Watanabe and Nakamura [62] gave an $O(n(n+m)^2)$ time two stage sequential algorithm for triconnectivity augmentation. Later, Hsu and Ramachandran [63] developed a linear-time algorithm for this problem.

Another Research Gap: In both the attempts, for triconnectivity augmentation, the associated 3-block tree is recomputed at each iteration of the algorithm, leading to a quite complicated combinatorial analysis. This calls for a simple triconnectivity augmentation algorithm avoiding the recomputation of the associated tree with elegant combinatorial analysis. This is the second motivation for our work.

Does 'tree-like' Structure Exist for General Graphs: Having seen the existence and importance of the *associated* tree in optimum bi(tri)connectivity augmentation, a natural question is to construct a *tree-like* structure for every k-connected graph and design a lower bound similar to the 3-block tree. It turns out that beyond 2-connected graphs, the notion of *tree-like* structure in general is not clear. For example, even for 3-connected graphs the existence of such a tree is not clear as described in [64]. Hsu [64] presented a different lower bound by understanding the structure of vertex separators of 3-connected graphs and an algorithm for four connecting a 3-connected graph and the algorithm in [64] precisely augments the number which equals the new lower bound and hence, the new lower bound is tight for 4-connectivity augmentation of 3-connected graphs. The algorithm in [64] runs in $O(n.\alpha(m,n)+m)$, where $\alpha(m,n)$ is the inverse Ackermann function. A different lower bound analysis using fragments and shredders to $(k+1)$-vertex connect a given k-connected graph is proposed by Jordan in [65]. For fixed k, he presented in [65] a polynomial-time algorithm to $(k+1)$-vertex connect a given k-connected graph and the run time of the algorithm is $O(n^6)$. A new characterization of a special graph, namely *independence free graphs* is also given in [65]. It is also mentioned in [65] that for all k, k-connectivity augmentation of an independence free graph can be obtained using the algorithm presented in [65]. For a graph G with arbitrary connectivity, he presented a *min-max* formula involving the number of edges to be augmented to G so that G is k-vertex connected. In what follows, the combinatorial analysis and the complexity of the vertex connectivity augmentation in general

were one of the most challenging open questions of this area. In other words, whether the decision version of the problem, is in P or NP-complete was open for a long time in the literature. Only recently, Vegh in [57] presented a polynomial-time algorithm for optimum $(k+1)$-connectivity augmentation of k-connected graphs.

While the construction of associated tree for k-connected graphs in general is not clear, it yields an optimum connectivity augmentation for 1-connected and biconnected graphs. It is natural to ask what other special graph classes can be augmented with associated tree based approach. It also gives an idea of the existence of $(k+1)$-block tree for k-connected graphs. It is interesting and tempting to analyze this observation which might yield an improved algorithm for connectivity augmentation of k-connected graphs over the algorithm reported in [57]. This motivates us to find a framework using which several special graph classes can be augmented optimally.

Other Connectivity Augmentations: Analogous to vertex connectivity augmentation, edge connectivity augmentation is another interesting problem. One version of the problem avoids adding parallel edges and hence focuses on simplicity preserving edge connectivity augmentation and other version allows multiple edges between a pair of vertices. The decision version of simplicity preserving edge connectivity augmentation(SPEA) is shown to be NP-complete in [66]. On the other hand, in [66] it is mentioned that SPEA can be solved in time $O(n^4)$ for fixed k. The other version of the problem was introduced in [58] and a linear-time algorithm is proposed for 2-edge connectivity augmentation. The general k-edge connectivity augmentation problem allowing multiple edges between a pair of vertices was solved by Watanabe and Nakamura [67]. A different approach to the same problem was given by Cai and Sun [68]. The key technique, *edge-splitting* is used in [68] and is proposed by Mader [69]. Frank in [70] showed that the local edge connectivity augmentation, a generalization of edge connectivity augmentation can be solved in $O(n^3 m \log(n^2/m))$. The local edge connectivity augmentation finds a minimum set of edges such that when it is augmented to the given graph, there are at least k edge disjoint paths between every pair of vertices.

The other basic augmentation problems namely to make a digraph k-vertex connected [71] and a digraph k-edge connected [70] are shown to be polynomial-time solvable. A digraph is called k-vertex/edge connected if it remains strongly connected by removal of any $(k-1)$ edges/vertices. The connectivity augmentation in digraphs finds a mini-

mum set of edges to be added to a given digraph such that the augmented digraph meets the prescribed connectivity. Frank in [70] pointed out that the edge connectivity augmentation in digraph can be solved in $O(n^3 m \log(n^2/m))$. Frank's survey [72] is an excellent source for more results on connectivity augmentation.

Interestingly, these problems are also of practical interest and are well motivated from the study of network reliability and fault-tolerant computing[73, 74, 75]. Constructing networks that are 'survivable' with respect to node/link failure is considered to be an important problem in the study of network reliability. In particular, we are interested in providing at least r node/link disjoint paths between every pair of nodes in the network. Connectivity augmentation focuses on finding a minimum number of edges whose augmentation guarantees at least r node/link disjoint paths between every pair of nodes in the network, r is a prescribed survivability of the network.

6.1.1 Our Contributions

Our goal is to explore the possibility of identifying a framework using which optimum connectivity augmentation can be done in several special graph classes. The two important issues to be considered in designing such a framework are: it must avoid the recomputation of the associated tree and it must yield an elegant algorithm with simple combinatorial analysis. It is important to identify such a framework which might give an handle on the existence of the associated trees for k-connected graphs in general. It is interesting and tempting to analyze this observation which might yield an improvement for connectivity augmentation in k-connected graphs over the algorithm reported in [57].

- Given a connected undirected unweighted graph, our framework first constructs the associated tree by understanding the structural decomposition of graphs with respect to minimum vertex separators. For example, given a 1-connected graph G, we construct the associated biconnected component tree where each node is either a cut vertex or a biconnected component of G. The highlights of this tree are it is unique and it captures all minimum vertex separators in the given graph. In all four augmentations reported in this work, we focus our combinatorial analysis on the associated tree to obtain an optimum connectivity augmentation set. As a first step, we propose an equivalence relation on the set of leaves of the tree. The set of equivalence classes is our data structure, which is precisely a partition

of the set of leaves of the associated tree. This partition determines the edge to be augmented to the graph at each iteration. Moreover, this partition guarantees a recursive sub-problem and it is sufficient to maintain this partition for finding an optimum connectivity augmentation set in graphs.
We report four applications of this data structure: two applications in this chapter and the other two in subsequent chapters. This chapter discusses biconnectivity augmentation of 1-connected graphs using biconnected component trees and triconnectivity augmentation of 2-connected graphs using 3-block trees. $(k+1)$-connectivity augmentation of k-trees and k-connected chordal graphs using minimum vertex separator trees and clique separator trees, respectively are discussed in the next two chapters. For each of the augmentations reported here, we first discuss the construction of the associated tree, followed by lower bound results for optimum connectivity augmentation. Based on the new framework we provide a new proof of tightness and an elegant algorithm for connectivity augmentation. As far as run-time analysis is concerned, chordal connectivity augmentation runs in cubic time and other augmentations run in linear time.

- It is important to compare our work with the results of [58, 60]. Given a 1-connected graph G, in [58, 60] a *block tree* T is constructed from G. It is also mentioned that an optimum biconnectivity augmentation for G can be obtained from T. An iterative algorithm in [60] identifies the edge to be added between a pair of leaves in T. Once an edge is added into T, it again computes the block tree. Since the addition of an edge creates a biconnected component and to get a tree after each iteration, algorithm in [60] recomputes block tree by dynamically maintaining the list of cut-vertices and the set of biconnected components. Recomputing the block tree and maintaining the additional information about cut-vertices makes the implementation of this algorithm a challenging task. Though [61] avoids the recomputation of the tree, it demands a rooted normalized tree. Moreover, the underlying combinatorial analysis is quite involved. In contrast, our algorithm avoids recomputing the tree using our new data structure. This new approach yields a simple algorithm for finding an optimum biconnectivity augmenting set. Our novelty is in the data structure that yields an elegant linear-time algorithm with simple combinatorial analysis. We also present a new algorithm for triconnectivity augmentation of biconnected graphs using our new data structure.

6.2 Biconnectivity Augmentation in Trees: A New Approach

Given a tree, this section presents a new approach to find an optimum augmenting set which makes the tree biconnected. We first discuss the lower bound on the size of optimum augmenting set, followed by a new proof of tightness. In this proof, we

identify an equivalence relation on the set of leaves of the tree. Using the partition of the set of leaves, namely, the set of equivalence classes we describe an approach to find an optimum biconnectivity augmenting set. We also show that it is sufficient to maintain this partition at each iteration of the algorithm. This new framework also guarantees a tree at each iteration and hence we obtain a recursive sub-problem efficiently. It is now natural to extend tree augmentation approach to augmentation in graphs which have tree like structure. Since trees are a subclass of 1-connected graphs, a natural extension is to study augmentation in 1-connected graphs by representing them using associated trees. The standard approach in the literature [58, 60] for biconnectivity augmentation of a 1-connected graph is to maintain a tree, namely, a block tree that captures the biconnected components, cut vertices, and bridges of a 1-connected graph. After the choice of an edge to be added is made, a new tree is computed, and the procedure stops when the tree becomes a single node. We work with biconnected component trees which are similar to block trees to find an augmenting set using our proposed framework. Our framework avoids recomputation of the associated tree at each iteration and this approach is fundamentally different from the results reported in [58, 60]. We also present a new algorithm for triconnectivity augmentation of biconnected graphs using our new data structure.

6.2.1 Lower Bound on Biconnectivity Augmentation in trees

Given a tree T we now present the lower bound on optimum biconnectivity augmenting set. It is a well-known fact that in any 2-connected graph, for any pair of vertices, there exists two vertex disjoint paths between them. This fact is useful in determining the lower bound on the optimum biconnectivity augmenting set. Let l denote the number of leaves in T. Clearly, to biconnect T we must augment at least $\lceil \frac{l}{2} \rceil$ edges. Another lower bound is due to the number of components created by removing a cut vertex of T. Note that the number of components created by removing a cut vertex x of T is precisely the degree of x in T. This shows that in any biconnectivity augmentation of T, for each cut vertex x, one must find at least $deg_T(x) - 1$ new edges in the augmenting set. Therefore, we must augment at least $\Delta(T) - 1$ edges to biconnect T. Therefore, by combining the two lower bounds, the number of edges to biconnect T is at least $\max\{\lceil \frac{l}{2} \rceil, \Delta(T) - 1\}$.

This lower bound is indeed tight as shown in [58, 60].

In the next section, we present a new proof of tightness. In this proof, we identify an equivalence relation on the set of leaves of the tree, and show that adding edges among appropriately chosen leaf pairs naturally results in a recursive sub-problem in which the lower bound value is one less. The main contribution here is the identification of the equivalence relation which consequently guarantees an easy construction of the recursive sub-problem. The equivalence relation and therefore the set of equivalence classes yield an elegant algorithm to compute $E_{min}(G)$, an optimum biconnectivity augmenting set. This approach is fundamentally different from the results presented in [60, 61]. We present an algorithm by focusing our combinatorial arguments on the set of equivalence classes. We further prove that the number of edges augmented by our algorithm is precisely the lower bound mentioned in this section.

6.2.2 An Equivalence Relation for Connectivity Augmentation

We now define a relation R on the set $L(T) = \{x_1, x_2, \ldots, x_l\}$ of leaves in T. Leaf x is related to leaf y, $y \neq x$ if there exists at most one vertex of degree at least 3 in P_{xy} and it is written as xRy. If x is not related to y in R, then it is written as $x\tilde{R}y$. In other words, the path P_{xy} contains at least two vertices z and z' such that $deg(z) \geq 3$ and $deg(z') \geq 3$.

Lemma 6.2.1 *R is an equivalence relation.*

Proof Clearly, the path P_{xx} contains no vertex of degree at least 3 and hence xRx. Moreover, this is true for all $x \in L(T)$. This implies that R is reflexive. Also, if xRy then it follows yRx as T is an undirected tree. Clearly, R is symmetric. To prove that R is transitive, for $x, y, w \in L(T)$, if xRy and yRw, we need to prove that xRw. Suppose $x\tilde{R}w$. This implies that the path P_{xw} contains at least two vertices z and z' such that $deg(z) \geq 3$ and $deg(z') \geq 3$. Since $x, y, w \in L(T)$, it follows that either the path P_{xy} or the path P_{yw} contains at least two vertices of degree at least 3. This implies that either $x\tilde{R}y$ or $y\tilde{R}w$. However, this is a contradiction to the given fact that xRy and yRw. Hence our assumption that $x\tilde{R}w$ is wrong. What follows is that the path P_{xw} contains

at most one vertex z such that $deg(z) \geq 3$. Therefore, xRw. Hence R is transitive. Therefore, R is an equivalence relation. $\square$

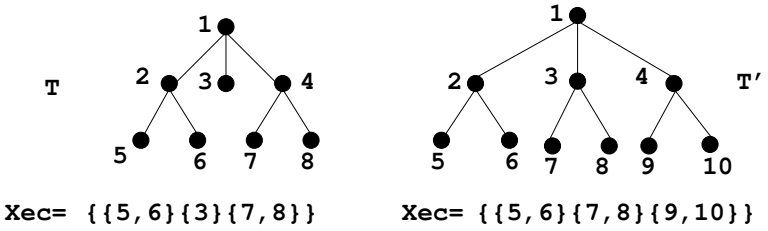


Figure 6.1: Examples Illustrating Equivalence Classes

Since R is an equivalence relation, R induces a set X_{ec} of equivalence classes on the set $L(T)$ of leaves in T. An example is illustrated in Figure 6.1. The following fact highlights a structural property of each equivalence class in X_{ec}.

Fact: *Each equivalence class is associated with a unique vertex (representative) of degree at least 3 in* T.

By the definition of our relation, leaf x and leaf y are related if there exists at most one vertex z of degree at least 3 in P_{xy}. This implies that z is the first vertex of degree at least 3 in P_{xy} and every other vertex in P_{xz} and P_{yz} is of degree at most 2 in T. Since R partitions the set of leaves into set of equivalence classes, we observe that, for each equivalence class $X \in X_{ec}$ there is an associated unique vertex, denoted by $w(X)$ such that $w(X)$ is the nearest vertex of degree at least 3 on the path from each element of X. We refer to $w(X)$ as the representative associated with X.

Before we present our algorithm we highlight one more fact which describes a special tree. If the tree T is such that T has exactly one vertex of degree at least 3, then by the definition of R, we get exactly one equivalence class containing all the leaves in T. The tree in this case is a star like tree. We call such a special tree as *star*.

A Generic Approach to Augmentation Algorithm: To decide upon the edge to be augmented at each iteration, we perform the following: from the set of equivalence classes, we identify two equivalence classes X and Y such that degree of its representatives are maximum and second maximum, add the edge $\{u, v\}$, $u \in X$ and $v \in Y$ and update the set of equivalence classes. The tail condition of this procedure is when there is exactly one equivalence class and we know from our earlier discussion that there is exactly one special tree namely *star* and augmentation for *star* is done separately.

6.2.3 Augmentation using Equivalence Classes

In this section, we present our linear-time algorithm to find an optimum biconnectivity augmenting sets in trees. An example illustrating the trace of the algorithm is also given. Let $X_{ec} = \{X_1, \ldots, X_r\}$ denote the set of equivalence classes with $w(X_i)$ being the associated representative of $X_i \in X_{ec}$. Let $M_0 = \max\{\lceil \frac{l}{2} \rceil, \Delta(T) - 1\}$. We use $\Delta(T)$ and Δ interchangeably and the tree to which it is associated will be clear from the context. Let Δ^i denote the value of $\Delta(T)$ at the end of i-th iteration of the algorithm. Similarly, l_i denotes the number of leaves in the tree at the end of the i-th iteration. At the end of i-th iteration of the algorithm, let $M_i = \max\{\lceil \frac{l_i}{2} \rceil, \Delta^i - 1\}$. The following key lemma is presented in [60] and this key observation leads to an optimum algorithm reported in [60]. We present our key observation in Lemma 6.2.3 and this observation yields an elegant combinatorial analysis and an algorithm for finding an optimum connectivity augmentation set.

Lemma 6.2.2 *[60] Let T be a tree with $l \geq 3$. If $\Delta(T) > \lceil \frac{l}{2} \rceil$ then there exists at most two cut-vertices $v_1, v_2 \in V(T)$ whose degree is $\Delta(T)$.*

We observe the following key result which is used in the proof of Lemma 6.2.4

Lemma 6.2.3 *Let T be a tree with $l \geq 3$ and $z \in V(T)$. If $deg(z) = \Delta > \lceil \frac{l}{2} \rceil$, then there exists an equivalence class $X \in X_{ec}$ such that $w(X) = z$.*

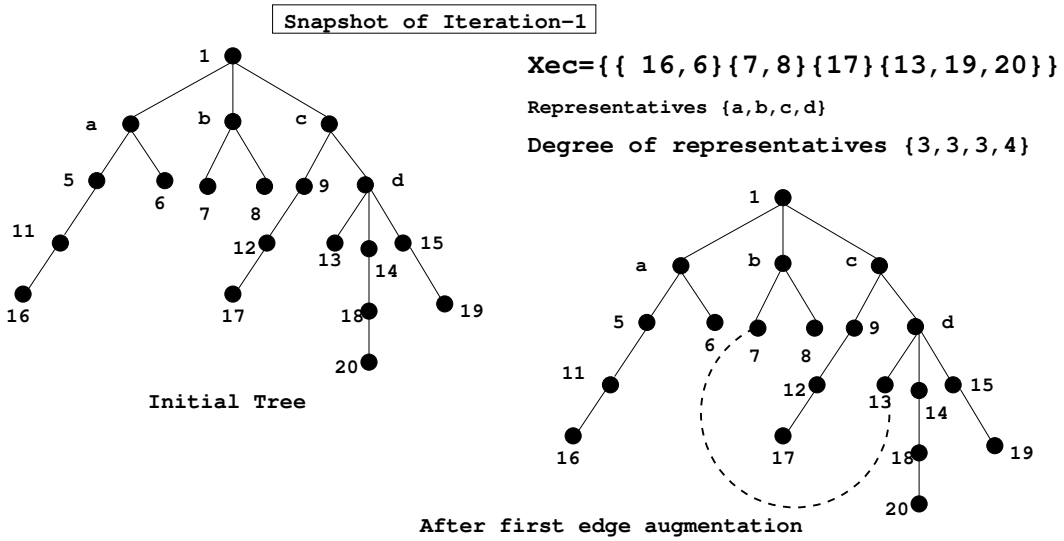
Proof Suppose there does not exist X such that $w(X) = z$. Then there are at least two leaves in each of the trees in the forest obtained by removing z. Also, by our hypothesis each component is neither an isolated vertex nor a path in $T \setminus \{z\}$. This implies that each component is a tree with at least two leaves and each of these leaves, except the $N_T(z)$ is indeed a leaf in T. Since $deg(z) = \Delta > \lceil \frac{l}{2} \rceil$, it follows that the number of leaves is at least $2\Delta > l$. A contradiction. Therefore, there exists an equivalence class $X \in X_{ec}$ such that $w(X) = z$. $\square$

Algorithm 1 Biconnectivity Augmentation in Trees: *tree-augment(Tree T)*

if there are exactly two leaves x and y **then**
  Add the edge $\{x, y\}$ and return the biconnected graph
else
  Compute the set X_{ec} of equivalence classes
  if $|X_{ec}| > 1$ **then**
    /* T is not a star */
    Y_{ec}=non-star-augment(X_{ec})
    star-augment(Y_{ec})
  else
    /*T is a star */
    star-augment(X_{ec})
  end if
end if

Algorithm 2 Biconnectivity Augmentation in stars: *star-augment(eclass-list X_{ec})*

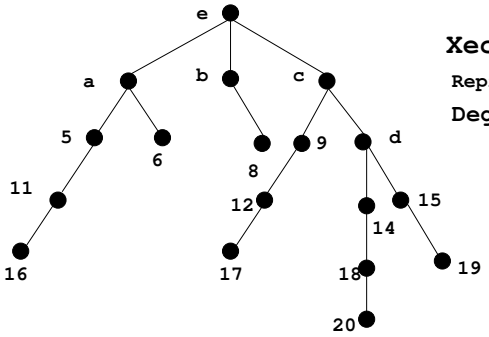
Let $X = \{x_1, \ldots, x_l\}$ be the set of leaves in T such that $X \in X_{ec}$
Add $|X| - 1$ edges to T, i.e., add $\{\{x_i, x_{i+1}\} \mid 1 \leq i \leq |X| - 1, x_i \in X\}$.



Algorithm 3 Biconnectivity Augmentation in non-star Trees: *non-star-augment(eclass-list X_{ec})*

1: **while** $|X_{ec}| \geq 2$ **do**
2: Let X_1 and X_2 are two equivalence classes in X_{ec} such that $deg(w(X_1)) \geq deg(w(X_2)) \geq deg(w(X_i)), i \geq 2$
3: Add the edge $\{x, y\}, x \in X_1, y \in X_2$. Remove x from X_1 and y from X_2.
4: Remove the path from x to $w(X_1)$ and y to $w(X_2)$ from T /* This yields a tree after augmentation */
5: Update X_{ec}
6: **end while**
7: **return** X_{ec}/*X_{ec} has single equivalence class and the associated tree is star */

Snapshot of Iteration-2

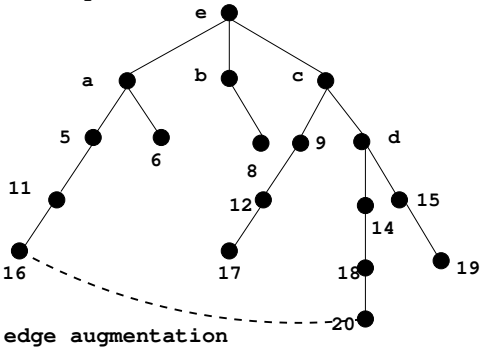


Tree after iteration-1

Xec={{ 16,6}{8}{17}{19,20}}

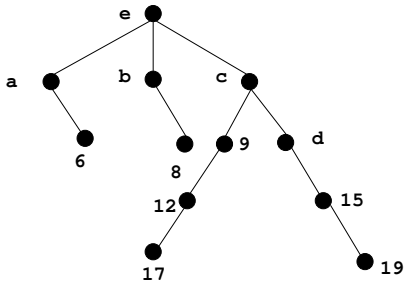
Representatives{a,e,c,d}

Degree of representatives {3,3,3,3}



After second edge augmentation

Snapshot of Iteration-3

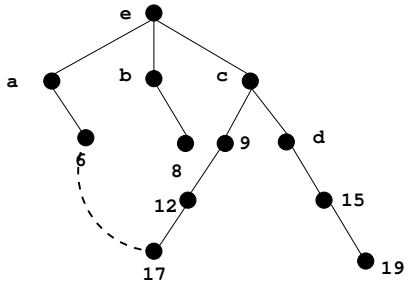


Tree after iteration-2

After Merging

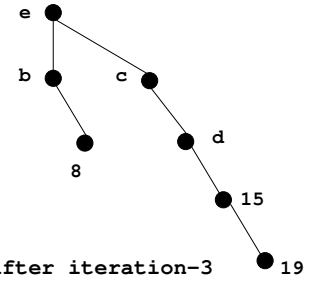
Xec={{ 6,8}{17,19}}

Representatives {e,c} with Degree {3,3}

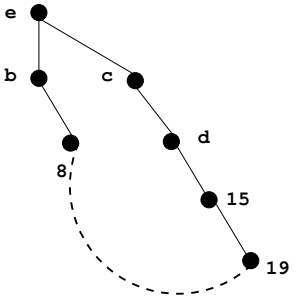


After third edge augmentation

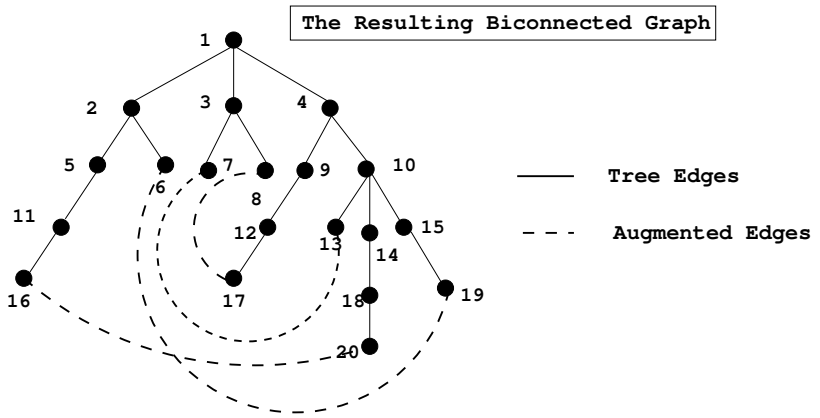
Snapshot of Iteration-4



Tree after iteration-3



After fourth edge augmentation



Lemma 6.2.4 *Let T be a tree such that T is not a star. Then $M_1 = M_0 - 1$.*

Proof Given that T is a tree such that T is not a star implies that $|X_{ec}| \geq 2$. Since, the maximum degree does not increase from one iteration to the next, and the number of leaves strictly reduces, clearly, $M_1 \leq M_0$. We prove that $M_1 = M_0 - 1$ by contradiction. Suppose $M_1 \neq M_0 - 1$. This implies that $M_1 = M_0$. We know that $l_1 = l_0 - 2$. Since $M_1 = M_0$ and $l_1 = l_0 - 2$ it must be the case that $\Delta^1 - 1 > \lceil \frac{l_1}{2} \rceil$. We prove this observation by contradiction. Suppose, $\Delta^1 - 1 \leq \lceil \frac{l_1}{2} \rceil = \lceil \frac{l_0}{2} \rceil - 1$. Consequently, $M_1 = \max\{\Delta^1 - 1, \lceil \frac{l_1}{2} \rceil\} = \lceil \frac{l_0}{2} \rceil - 1$. Further, $M_1 = M_0$ implies that $\lceil \frac{l_0}{2} \rceil - 1 = M_1 = M_0 \geq \lceil \frac{l_0}{2} \rceil$, and this is a contradiction. Therefore, our observation that $\Delta^1 - 1 > \lceil \frac{l_1}{2} \rceil$ is true. Further, since $l_1 = l_0 - 2$ and $M_1 = M_0$, it must be that $\Delta^1 = \Delta^0$. Therefore, $\Delta^0 - 1 > \lceil \frac{l_0}{2} \rceil - 1$ and hence $\Delta^0 > \lceil \frac{l_0}{2} \rceil$. Therefore, now applying Lemma 6.2.2, we know that there can be at most two vertices $v_1, v_2 \in V(T)$ of degree Δ^0. Further, from Lemma 6.2.3, a cut-vertex of maximum degree is the representative associated with an equivalence class. Since the biconnectivity augmentation adds an edge between $x \in X_1$ and $y \in X_2$ such that $deg(w(X_1)) \geq deg(w(X_2)) \geq deg(w(X_i)), 2 \leq i \leq r$, it follows that the degrees of the associated representatives also reduce by one. Since the maximum degree vertices have degree more than $\lceil \frac{l}{2} \rceil$, there are at most two of them (by Lemma 6.2.2), both are associated representatives of two equivalence classes (by Lemma 6.2.3), and the algorithm adds an edge between two vertices in these two equivalence classes, it follows that the maximum degree reduces by 1. That is, $\Delta^1 = \Delta^0 - 1$ and this contradicts our earlier conclusion that $\Delta^1 = \Delta^0$. Therefore, our starting premise, $M_1 = M_0$ is wrong, $M_1 = M_0 - 1$. $\square$

Theorem 6.2.5 *Let T be a tree. The minimum biconnectivity augmentation of T uses $M_0 = \max\{\lceil \frac{l}{2} \rceil, \Delta(T) - 1\}$ edges.*

Proof The algorithm guarantees that at the end of each iteration we always have a tree. Further, we know from Lemma 6.2.4 that if $|X_{ec}| \geq 2$, then $M_1 = M_0 - 1$. Therefore, let us assume that *while-loop* of the function *non-star-augment* is called p times, after which *star-augment* is called once. Then $M_p = M_0 - p$. In other words, in the tree obtained after p calls to *while-loop* of *non-star-augment* there is a single equivalence class. In such a situation, we know from our earlier discussion that the resulting tree obtained from *non-star-augment* is a star. We know that for star $\Delta^p = l_p$, and by definition $M_p = \Delta^p - 1$. *star-augment* function biconnects this tree using M_p edges. Therefore, the total number of edges added is $M_p + p = M_0 - p + p = M_0$. It is also easy to see by induction on p that the set of edges added is a biconnectivity augmentation set. The base case is when $p = 0$, and clearly, *star-augment* biconnects the tree T. After the first iteration, the resulting tree goes through a strictly smaller number of iterations, and we assume by induction on the number of iterations that the algorithm returns a biconnectivity augmentation set using $M_0 - 1$ edges. The first iteration counts one more edge that ensures an additional path between the unaccounted vertices of degree at most two, thus guaranteeing biconnectivity. Note that the unaccounted vertices are two paths in the tree each originating at a distinct leaf from two distinct equivalence classes, namely X_1 and X_2. □

6.2.4 A Linear-time Implementation of *non-star-augment()* using a Novel Data Structure

The important steps to be analyzed in our algorithm are computing the set of equivalence classes, finding two equivalence classes such that the degree of its representatives are maximum and second maximum, and updating of equivalence classes after edge additions. We below mention possible situations that may arise on execution of lines (3)-(5) of Algorithm 3 and the specific tasks to be taken in updating the set of equivalence classes.

Table 6.1: Operations Defined on *Equivalence-Class-List* ADT

Set-up-eclass()	This creates the set of equivalence classes from the tree. This is the first method to be called to populate the data structure
Locate(L, w)	Return from L, the location of equivalence class whose representative is w
Insert(L, pos, l)	Insert the leaf node l into the equivalence class whose position in L is pos and return the updated list L
Retrieve(L, pos)	Return from L, an element of the equivalence class whose position is pos
Parent-Representative-Indicator(L, x)	Return from L, the *parent-indicator* of leaf x

- For an equivalence class X, if $deg(w(X)) = 2$ and $|X| = 1$ then by definition, $w(X)$ is no longer a representative vertex of X. The *update* in this case is the identification of the new representative of X. The update is done by identifying a nearest vertex z of degree at least 3 from $w(X)$. Since $deg(z) \geq 3$, there must be an equivalence class Y (possibly empty) such that $w(Y) = z$. We say that X *merges* with the equivalence class Y. To identify such a z efficiently, we maintain an additional information at each leaf x to ensure that the search for z does not happen on the path from $w(X)$ to x. We call this additional information as *parent-indicator* for each leaf x in T. For a leaf $x \in X$, the *parent-indicator* of x is a vertex x' such that $x' \in N_G(w(X))$ and $x' \in P_{xw(X)}$. The purpose of *parent-indicator* is that the search for z must avoid $P_{x'x}$ as every vertex in $P_{x'x}$ is of degree two.
- The other possible situations are $|X| = 0$ or $|X| \geq 2$ or $|X| = 1$ and $deg(w(X)) \geq 3$. In this case, the *update* must reorganize the equivalence classes based on the degree of the representatives.

We propose an abstract data type (ADT) to *maintain* the set of equivalence classes. We call the ADT as *Equivalence-Class-List* and using the operations listed in Table 6.1, we present a linear-time implementation of the above reorganization steps. We use two basic data types namely, *eclass-list* to refer the data type of X_{ec} and *node* to refer the data type of a vertex x. Let L be an object of type *eclass-list*.

Data Structures Used: The main data structure which is populated by *Set-up-eclass()* is a table of records, which we call *table-eclass*. For each vertex of degree at least 3 in T there is a record in *table-eclass*. Each record has three fields, the label of a vertex w of degree at least 3, the degree of w in T, and the subset of leaves in the equivalence class associated with w. The method *Set-up-eclass()* fills entries in *table-eclass* by performing Depth First Search(DFS) on T. During the DFS, for each vertex $w, deg(w) \geq 3$,

we find the equivalence class of w. We also store the *parent-indicator* of each element in the equivalence class associated with vertex w (i.e. for each leaf in T). Since each vertex is visited at most twice during the DFS, construction of *table-eclass* takes at most $2|E(T)|$ and hence $O(n)$ time. With this table, ADT methods *Insert*, *Retrieve*, and *Parent-Representative-Indicator* can be implemented in constant time.

To index *table-eclass* to locate the record labelled w, in constant time, we use *index-table[ ]* array to store the position of w in *table-eclass* table. This implements the ADT method *Locate* in constant time.

The subroutines *get-top-two-representative* and *update-eclass* implement lines (2) and (5), respectively of Algorithm 3. These subroutines make use two additional data structures. The data structure *initial-active-eclass[ ]* array contains vertices x in non-increasing order of their degrees such that there is a non-empty equivalence class associated with x. This can be achieved by running radix sort on the associated set and it runs in $O(n)$ time. *sorted-vertex[i]* which contains a list of vertices x of degree i in T. Initially, *sorted-vertex[i]* is filled using elements of *initial-active-eclass[ ]*. We present the pseudo code of Algorithm 3 in procedure *MAIN*.

Run-time Analysis: The overall time complexity of *non-star-augment* algorithm is given as follows: Depth First Search(DFS) on the given tree to fill *table-eclass* incurs $O(n)$ time. Running radix sort and creating an entry in *sorted-vertex[ ]* takes $O(n)$ time. With supporting data structures, we incur constant time effort for the subroutine *get-top-two-representative*. For the subroutine *update-eclass*, we are analyzing the total effort involved over all iterations. If there is a *merging* then to identify the nearest vertex of degree at least 3, the total time spent for the above operation over all iterations is $O(n)$, as we visit each vertex exactly once. This is possible because of *parent-indicator* information stored at each leaf. If there is no *merging* then we incur constant time effort to reorganize the set X_{ec}. As per our algorithm, each vertex of degree at least 3 is visited at most the size of equivalence class associated with that vertex. Since the sum of the size of all equivalence classes is at most the sum of degrees in the tree, the time spent in identifying the augmentation set is $O(n)$. When the tree is star, *star-augment* incurs an additional $O(n)$ time. Therefore, the total time complexity of *tree-augment* algorithm is $O(n)$, linear in the input size.

Procedure-MAIN()

1: get-top-two-representative()
2: edge-addition() /* this subroutine incurs constant time */
3: remove-path-from-leaf-to-representative()/* this implements line(4) of Algorithm 3 , total time spent over all iterations is $O(n)$ */
4: update-eclass()

Procedure 1: get-top-two-representative()

1: /* max1=initial-active-eclass[0], max2=initial-active-eclass[1] This subroutine incurs constant time*/
2: Remove the first element from the list associated with *sorted-vertex[max1]*, say x
3: max1-pos-table=locate(x)
4: Retrieve(L,max1-pos-table)
5: Decrement the $deg(x)$ by one in the table
6: Remove the first element from the list associated with *sorted-vertex[max2]*, say y
7: max2-pos-table=locate(y)
8: Retrieve(L,max2-pos-table)
9: Decrement the $deg(y)$ by one in the table

Procedure 2: update-eclass(L): eclass-list

1: /* This procedure updates X_{ec} after edge addition. Here we outline the specific tasks to be taken during update X_{ec}. $X_1, X_2 \in X_{ec}$ such that $deg(w(X_1)) = max1$ and $deg(w(X_2)) = max2$. Every line incurs constant time */
2: We first check whether X_1 or X_2 merge with other element in X_{ec}. Merging happens, if $|X_i| = 1$ and $deg(w(X_i)) = 2, i \in \{1, 2\}$.
3: If **merging=true** (for our discussion assume X_1 merges), perform the following 4 steps
4: Follow the path P from $w(X_1)$ till we hit the first vertex w' of degree at least 3 such that P does not contain *parent-indicator* of leaf $x \in X_1$ /* For this step, the total time spent over all iterations is $O(n)$
5: Merge X_1 with the equivalence class associated with w'.
6: Make an entry for w' in *sorted-vertex[ ]*, if not exists. Reset *max1* and *max2* using *sorted-vertex[ ]* and *initial-active-eclass[ ]*
7: Update *parent-indicator* of x.
8: If **merging=false**, reset *max1* and *max2* using *sorted-vertex[ ]* and *initial-active-eclass[ ]*

6.3 An Application of Equivalence Classes: Biconnectivity Augmentation

In this section, we first generalize tree augmentation results and present an augmentation in 1-connected graphs. As mentioned earlier, the approach is to represent a 1-connected graph by an appropriate tree, and find an optimum augmentation set using the results presented in Section 6.2.

The Associated tree Description: We represent the given 1-connected graph by a tree, namely, the biconnected component tree. A biconnected component is a maximal 2-connected subgraph of a given graph. A biconnected component tree T is a tree constructed from the given graph G as follows: each vertex in T denotes either a biconnected component of G or a cut-vertex of G. For a vertex $x \in V(T)$, the associated label $label(x) = \{c\}$, c is a cut-vertex in G or $label(x) = S$, $S \subseteq V(G)$ such that $G[S]$ is a maximal 2-connected subgraph in G. $V(T) = B \cup B'$ where $B = \{x \mid label(x)$ is a biconnected component in $G\}$ and $B' = \{x \mid label(x)$ is a cut-vertex in $G\}$. The adjacency between a pair of vertices in T is defined as follows: for $x, y \in V(T)$, $\{x, y\} \in E(T)$ if one of the following is true (see Figure 6.2 for an illustration)

- $x \in B$ and $y \in B'$ such that $label(y) \subset label(x)$
- Both $x, y \in B'$ and there is a $\{c, c'\}$ cut-edge in G such that $label(x) = \{c\}$ and $label(y) = \{c'\}$
- $x \in B$ and $y \in B'$ such that $label(x)$ is a trivial biconnected component($|label(x)| = 1$) and there is a $\{u, v\}$ cut-edge in G such that $label(x) = \{u\}$ and $label(y) = \{v\}$.

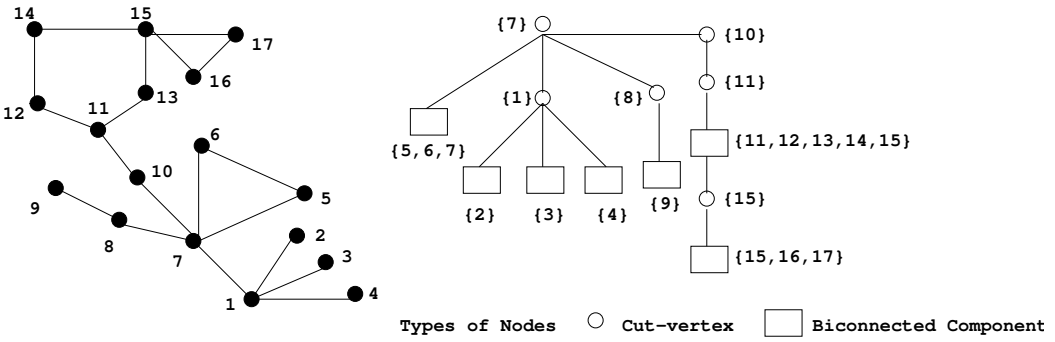


Figure 6.2: A 1-connected graph and the associated biconnected component tree

6.3.1 Lower Bound on Biconnectivity Augmentation in 1-connected Graphs

We use a well-known fact that in any 2-connected graph, for any pair of vertices, there exists two vertex disjoint paths between them. Let X denote the set of biconnected components having *exactly one* cut-vertex in G. Note that by our construction of T, each element of X corresponds to the *label* of a leaf in T. Clearly, to biconnect G we must augment at least $\left\lceil \frac{|X|}{2} \right\rceil$ edges and therefore we need at least $\lceil \frac{l}{2} \rceil$ edges, l denotes the number of leaves in T. Another lower bound is due to the number of components created by removing a cut vertex of G. Note the one-one correspondence between the number of components created by a cut vertex of G and the degree of a vertex $x \in B'$. This shows that in any biconnectivity augmentation of G, for each $x \in B'$, one must find at least $deg_T(x) - 1$ new edges in the augmenting set. Therefore, we must augment at least $\Delta_c(T) - 1$ edges to biconnect G, where $\Delta_c(T) = \max_{x \in B'} deg(x)$. Therefore, by combining the two lower bounds, the number of edges to biconnect G is at least $\max\{\lceil \frac{l}{2} \rceil, \Delta_c(T) - 1\}$. This lower bound is indeed tight as shown in [58, 60].

Note the similarity between this lower bound and the one presented in the previous section. $\Delta_c(T)$ appears here instead of $\Delta(T)$. Because of the similarity between the two lower bounds and the associated representation is a tree, all combinatorial analysis presented in the previous section can naturally be extended to biconnectivity augmentation in 1-connected graphs. Hence, we get a new proof of tightness which leads to a new linear-time algorithm for finding a biconnectivity augmentation set in a 1-connected graph. The equivalence relation, and therefore, the set of equivalence classes yields an elegant algorithm that completely avoids the recomputation of the associated tree at each iteration, unlike, the results presented in [59, 60].

6.3.2 Algorithm for Biconnectivity Augmentation in 1-connected Graphs

The Sketch of Algorithms 4 and 5: Compute the set X_{ec} of equivalence classes of biconnected component tree T. If $|X_{ec}| \geq 2$ then find an augmenting set using the subroutine *non-star-augment* of Section 6.2. The tail condition is when there is exactly one

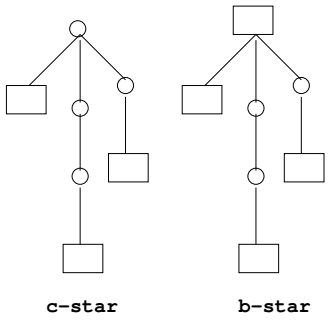


Figure 6.3: A c-star and a b-star

equivalence class and such a tree is a star shaped tree. In this case, we get two special trees depending on the label of the representative vertex z. We know that $label(z)$ is either a set containing a cut-vertex of G or a maximal 2-connected subgraph in G. If $label(z)$ is a set containing a cut-vertex of G, then we call T as a *c-star*. If $label(z)$ is a maximal 2-connected subgraph in G, then we call T as a *b-star*. An example is shown in Figure 6.3. An augmentation for tail condition is done separately. With this new approach, we simplify our biconnectivity augmentation algorithm by calling *non-star-augment* subroutine with biconnected component tree as the input.

Let $M_0 = \max\{\lceil \frac{l}{2} \rceil, \Delta_c(T) - 1\}$. Observe that $\Delta_c(T)$ appears in this expression

Algorithm 4 Augmentation in 1-connect-augment: *1-connect-augment(Graph G)*

Compute the biconnected component tree T of G
if there are exactly two leaves x and y in T **then**
  Add the edge $\{x, y\}$ and return the biconnected graph
else
  Compute equivalence classes $X_{ec} = \{X \mid$ X is an equivalence class with associated vertex w(X)$\}$
  if $|X_{ec}| > 1$ **then**
    `/* T is not a star-like tree */`
    Y_{ec}=non-star-augment(X_{ec}) `/* Call to non-star-augment() of Section 6.2, returns` Y_{ec} `*/`
    bc-star-augment(Y_{ec}) `/*` Y_{ec} `has exactly one equivalence class. The associated tree is a star */`
  else
    `/*`T `is a star-like tree */`
    bc-star-augment(X_{ec})
  end if
end if
For each edge $\{x, y\}$ added into T, add $\{u, v\}$ to G such that $u \in label(x)$ and $v \in label(y)$ and u and v are non-cut vertices in G

instead of $\Delta(T)$. With this observation, we see that Lemmas 6.2.2,6.2.3, and 6.2.4 are

true in biconnected component tree T as well. We only present a proof of Theorem 6.3.1.

Algorithm 5 Biconnectivity Augmentation in stars: *bc-star-augment(List-of-eclass X_{ec})*

Let $X = \{x_1, \ldots, x_l\}$ be the set of leaves in T such that $X \in X_{ec}$
if T is a c-star **then**
  `/* T is a star-like tree with a central vertex corresponding to a cut-vertex in G */`
  Add $|X| - 1$ edges to T, i.e. add $\{\{x_i, x_{i+1}\} \mid 1 \leq i \leq |X| - 1, x_i \in X\}$.
else
  `/* T is a star-like tree with a central vertex corresponding to a biconnected component in G */`
  if l is even **then**
    Add $\{\{x_i, x_{l-i+1}\} \mid 1 \leq i \leq \frac{l}{2}\}$ to T
  else
    Add $\{\{x_i, x_{(l-1)-i+1}\} \mid 1 \leq i \leq \frac{l-1}{2}\} \cup \{x_1, x_l\}$ to T
  end if
end if

Theorem 6.3.1 *Let T be a biconnected component tree of a 1-connected graph G. The minimum biconnectivity augmentation of G uses $M_0 = \max\{\lceil \frac{l}{2} \rceil, \Delta_c(T) - 1\}$ edges.*

Proof The algorithm guarantees that at the end of each iteration we always have a tree. Further, we know from Lemma 6.2.4 that if $|X_{ec}| \geq 2$, then $M_1 = M_0 - 1$. Therefore, let us assume that *while-loop* of the function *non-star-augment* of Section 6.2 is called p times, after which *bc-star-augment* is called once. Then $M_p = M_0 - p$. In other words, in the tree obtained after p calls to *while-loop* of *non-star-augment* there is a single equivalence class. If that equivalence class is associated with a cut-vertex of G then the resulting tree obtained from *non-star-augment* is a c-star. We know that for c-star $\Delta_c^p = l_p$, and by definition $M_p = \Delta_c^p - 1$. *bc-star-augment* function biconnects this tree using M_p edges. Therefore, the total number of edges added is $M_p + p = M_0 - p + p = M_0$. If that equivalence class is associated with a biconnected component of G then the resulting tree obtained from *non-star-augment* is a b-star. The total number of edges added in this case is $\left\lceil \frac{l-l_p}{2} \right\rceil + \left\lceil \frac{l_p}{2} \right\rceil = \lceil \frac{l}{2} \rceil$. It is also easy to see by induction on p that the set of edges added is a biconnectivity augmentation set. The base case is when $p = 0$, and clearly, *bc-star-augment* biconnects the graph G. After the first iteration, the resulting tree goes through a strictly smaller number of iterations,

and we assume by induction on the number of iterations that the algorithm returns a biconnectivity augmentation set using $M_0 - 1$ edges. The first iteration counts one more edge that ensures an additional path between the unaccounted vertices of degree at most two, thus guaranteeing biconnectivity. Note that the unaccounted vertices are two paths in the tree each originating at a distinct leaf from two distinct equivalence classes, namely X_1 and X_2. Hence the theorem.□

The overall time complexity of our algorithm is given as follows: Depth First Search(DFS) on the given graph can be used to compute the biconnected component tree in $O(n+m)$ time [76]. The number of nodes in the biconnected component tree is $O(n)$. We know from previous section that *non-star-augment* can be implemented in $O(n)$ time. Hence, the total time complexity of our algorithm is $O(n + m)$, linear in the input size.

6.4 Another Application of Equivalence Classes: Triconnectivity Augmentation

We present an elegant linear-time algorithm that finds a minimum set of edges whose augmentation to a 2-connected graph makes it 3-connected. Tutte in [30, 77] presented a structural characterization of 2-connected graphs with respect to vertex separators. He mentioned that a 2-connected graph can be decomposed into three sets, namely, a set of 2-sized vertex separators, a set of triconnected components, and a set of polygons. In [78], Hopcroft and Tarjan presented a linear-time algorithm to find such a decomposition from a 2-connected graph. Interestingly, one can construct a tree like graph from such a decomposition. Moreover, the tree obtained from such a decomposition is unique. Hsu and Ramachandran have shown in [79], a construction of such a tree, namely, a 3-block tree from a 2-connected graph. For the sake of completeness, we present here the construction of a 3-block tree from a 2-connected graph and the construction is as follows:

The Associated tree Description: Given a 2-connected graph G, the vertex set of a 3-block tree T, $V(T) = \{x \mid label(x)$ is a triconnected component (a maximal 3-connected subgraph) in G or a 2-sized vertex separator or a polygon or a vertex of

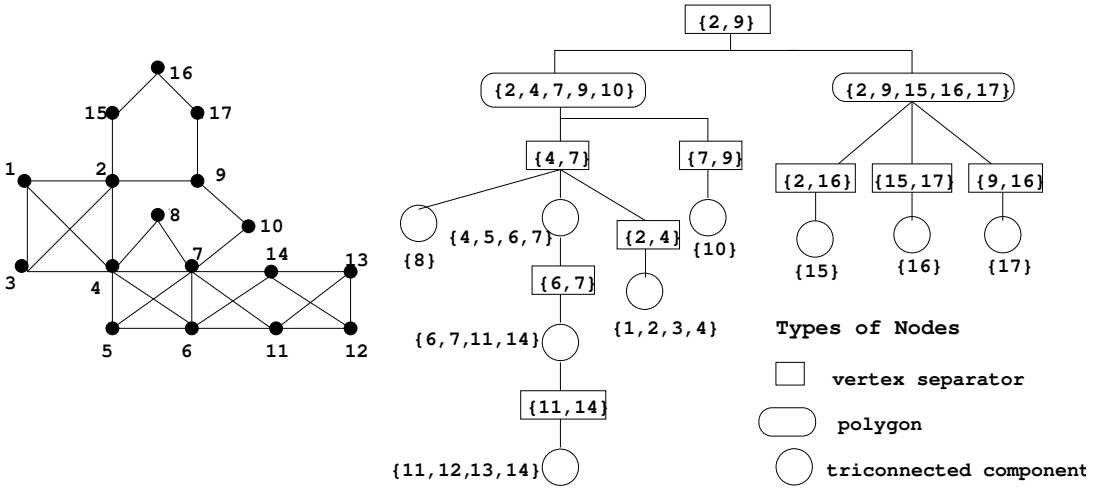


Figure 6.4: A 2-connected graph and the associated 3-block tree

degree 2$\}$. For convenience, we use the following notation. $V(T)$ consists of four kinds of vertices called σ vertices, π vertices, α vertices, and β vertices. We create a σ vertex for each 2-sized vertex separator such that components separated by the vertex separator is either a triconnected component or a polygon, a π vertex for each polygon, a α vertex for each vertex of degree 2, and a β vertex for each triconnected component. A vertex of degree 2 (α vertex) is the only trivial triconnected component. Note that the set of α vertices is a subset of the set of β vertices. The adjacency between a pair of vertices in $V(T)$ is defined as follows: for $x, y \in V(T)$, $\{x, y\} \in E(T)$, if one of the following is true (see Figure 6.4 for an illustration)

- $x \in \sigma$ and $y \in \beta$ and $label(x) \subset label(y)$.
- $x \in \sigma$ and $y \in \pi$ and $label(x) \subset label(y)$.
- $x \in \alpha$ and $y \in \sigma$ such that $label(y) = N_G(x)$.

Note that we do not create a σ vertex for each pair of non-adjacent vertices in a polygon (π vertex), we create a σ vertex for a pair in the polygon if it is involved in the *Tutte split*. More information about *Tutte split* and *Tutte merge* can be found in [77]. Let $\Delta_\sigma(T) = \max_{x \in \sigma} deg(x)$.

6.4.1 Lower Bound on Triconnectivity Augmentation in Biconnected graphs

Given a 2-connected graph G, we now present the lower bound on an optimum triconnectivity augmenting set. A well-known fact that in any 3-connected graph, for any pair of vertices, there exists three vertex disjoint joint paths between them. This fact is useful in determining the lower bound on the optimum triconnectivity augmenting set. Let X denote the set of triconnected components having *exactly one* 2-sized vertex separator in G. Note that by our construction of T, X corresponds to the set of leaves in 3-block tree T. Clearly, to triconnect G we must augment at least $\left\lceil \frac{|X|}{2} \right\rceil$ edges and therefore, we need at least $\lceil \frac{l}{2} \rceil$ edges, l denotes the number of leaves in T. Another lower bound is due to the number of components created by removing a vertex separator of size 2 in G. Note the one-one correspondence between the number of components created by a 2-sized vertex separator of G and the degree of a vertex $x \in \sigma$. This shows that in any triconnectivity augmentation of G, for each $x \in \sigma$, one must find $deg_T(x) - 1$ new edges in the augmenting set. Therefore, we must augment at least $\Delta_\sigma(T) - 1$ edges to triconnect G. Therefore, by combining the two lower bounds, the number of edges to triconnect G is at least $\max\{\lceil \frac{l}{2} \rceil, \Delta_\sigma(T) - 1\}$.

Note the similarity between this lower bound and the one presented in Section 6.2. $\Delta_\sigma(T)$ appears here instead of $\Delta(T)$. Because of the similarity between the two lower bounds and the associated representation is a tree, all combinatorial analysis presented in Section 6.2 can naturally be adapted for triconnectivity augmentation of 2-connected graphs. Hence, we get a new proof of tightness leading to a new linear-time algorithm for finding triconnectivity augmentation sets in 2-connected graphs. This approach is fundamentally different from the results presented in [62, 63].

6.4.2 Algorithm for Triconnectivity Augmentation in Biconnected graphs

The Sketch of Algorithms 6 and 7: Compute the set X_{ec} of equivalence classes of 3-block tree T. If $|X_{ec}| \geq 2$ then find an augmenting set using the subroutine *non-star-augment* of Section 6.2. The tail condition is when there is exactly one equivalence class

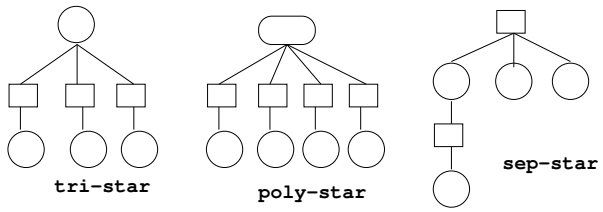


Figure 6.5: The three special star like trees, tri-star (if root is a triconnected component), poly-star (if root is a polygon) and sep-star (if root is a separator)

and such a tree is a star shaped tree. In this case, we get three special trees depending on the label of the representative vertex z. z can be a β (a triconnected component) or σ (a vertex separator of size 2) or π (a polygon) vertex of T and accordingly we call T as *tri-star*, *sep-star*, and *poly-star*, respectively. An example is given in Figure 6.5. An augmentation for tail condition is done separately. With this new approach, we simplify our triconnectivity augmentation algorithm by calling *non-star-augment* subroutine with 3-block tree as the input.

A proof similar to Theorem 6.3.1 can be given to prove that the number of edges

Algorithm 6 Augmentation in 2-connected graphs: *2-connect-augment(Graph G)*

```
Compute the 3-block tree T of G
if there are exactly two leaves x and y in T then
   Add the edge {x, y} and return the triconnected graph
else
   Compute equivalence classes X_ec = {X | X is an equivalence class with associated vertex w(X)}
   if |X_ec| > 1 then
      /* T is not a star-like tree */
      Y_ec=non-star-augment(X_ec)   /* Call to non-star-augment() of
      Section 6.2, returns Y_ec*/
      tail-star-augment(Y_ec)    /* Associated tree is a star with a
      single equivalence class */
   else
      /*T is a star-like tree */
      tail-star-augment(X_ec)
   end if
end if
For each edge {x, y} added into T, add {u, v} to G such that u ∈ label(x) and
v ∈ label(y) and u and v are elements of β vertex and not elements of σ vertex in G
```

augmented by our algorithm is $\max\{\lceil \frac{l}{2} \rceil, \Delta_\sigma(T) - 1\}$ and the resulting graph is 3-connected.

Algorithm 7 Triconnectivity Augmentation in stars: *tail-star-augment(List-of-eclass X_{ec})*

Let $X = \{x_1, \ldots, x_l\}$ be the set of leaves in T such that $X \in X_{ec}$
if T is a sep-star **then**
  `/* T is a star-like tree with a central vertex corresponding to a σ vertex */`
  Add $|X| - 1$ edges to T, i.e. add $\{\{x_i, x_{i+1}\} \mid 1 \leq i \leq |X| - 1, x_i \in X\}$.
else
  `/* T is a star-like tree with a central vertex corresponding to a π vertex or β vertex */`
  if l is even **then**
    Add $\{\{x_i, x_{l-i+1}\} \mid 1 \leq i \leq \frac{l}{2}\}$ to T
  else
    Add $\{\{x_i, x_{(l-1)-i+1}\} \mid 1 \leq i \leq \frac{l-1}{2}\} \cup \{x_1, x_l\}$ to T
  end if
end if

Theorem 6.4.1 *Let T be a 3-block tree of a biconnected graph G. The minimum triconnectivity augmentation of G uses $M_0 = \max\{\lceil \frac{l}{2} \rceil, \Delta_\sigma(T) - 1\}$ edges.*

Proof The algorithm guarantees that at the end of each iteration we always have a tree. Further, we know from Lemma 6.2.4 that if $|X_{ec}| \geq 2$, then $M_1 = M_0 - 1$. Therefore, let us assume that *while-loop* of the function *non-star-augment* of Section 6.2 is called p times, after which *tail-star-augment* is called once. Then $M_p = M_0 - p$. In other words, in the tree obtained after p calls to *while-loop* of *non-star-augment* there is a single equivalence class. If that equivalence class is associated with a σ vertex of G, then the resulting tree obtained from *non-star-augment* is a *sep-star*. We know that for *sep-star* $\Delta_\sigma^p = l_p$, and by definition $M_p = \Delta_\sigma^p - 1$. *tail-star-augment* function triconnects this tree using M_p edges. Therefore, the total number of edges added is $M_p + p = M_0 - p + p = M_0$. If that equivalence class is associated with a triconnected component or a polygon of G, then the resulting tree obtained from *non-star-augment* is a *tri-star* or *poly-star*. The total number of edges added in this case is $\left\lceil \frac{l-l_p}{2} \right\rceil + \left\lceil \frac{l_p}{2} \right\rceil = \lceil \frac{l}{2} \rceil$. It is also easy to see by induction on p that the set of edges added is a triconnectivity augmentation set. The base case is when $p = 0$, and clearly, *tail-star-augment* triconnects the graph G. After the first iteration, the resulting tree goes through a strictly smaller number of iterations, and we assume by induction on the number of iterations that the algorithm returns a triconnectivity augmentation set using $M_0 - 1$ edges. The first iteration counts one more edge that ensures an additional path between

the unaccounted vertices of degree at most two, thus guaranteeing triconnectivity. Note that the unaccounted vertices are two paths in the tree each originating at a distinct leaf from two distinct equivalence classes, namely X_1 and X_2. $\square$

From [78], we know that the associated 3-block tree of a 2-connected graph can be constructed in linear time. From Section 6.2, we know that a triconnectivity augmentation set can be obtained in linear time as well. Therefore, the overall time complexity to triconnect a biconnected graph is linear in the input size.

Concluding Remarks: In this chapter, we have presented a framework for finding an optimum connectivity augmentation set for various graph classes. From the associated tree of the given graph, our framework maintains a partition of the set of leaves which inturn guide us in finding an optimum augmentation set. We have also shown using three case studies that to find an optimum augmentation set it is sufficient to maintain this partition under edge additions. We have proposed a linear-time algorithm with elegant combinatorial analysis for bi(tri) connectivity augmentation. We demonstrate two more applications of this framework in the next chapter, namely, $(k+1)$-connectivity augmentation in k-trees and $(k+1)$-connectivity augmentation in k-connected chordal graphs.

CHAPTER 7

Connectivity Augmentation in Chordal Graphs

The focus of this chapter is to demonstrate the connectivity augmentation framework introduced in the last chapter in k-trees and k-connected chordal graphs. Biconnectivity augmentation of 1-connected graphs reported in the last chapter acts as a subroutine for connectivity augmentation in k-trees and triconnectivity augmentation of 2-connected graphs reported in the last chapter acts as a subroutine for connectivity augmentation in k-connected chordal graphs.

7.1 Augmentation in k-trees using Equivalence Classes

Given a k-tree G, we find a minimum set of edges whose augmentation to G makes it $(k+1)$-connected. A k-tree is defined recursively as follows: a $k+1$ clique is a k-tree, if G' is a k-tree, the graph $G = G' \cup \{v\}$ such that $N_G(v)$ is a k-clique in G' is a k-tree. Note that 1-trees are trees and k-trees can be seen as a natural generalization of trees. It is curious to see whether connectivity augmentation in trees can be extended to connectivity augmentation in k-trees. Interestingly, our methods naturally extend to connectivity augmentation in k-trees, and this does not seem to be easily feasible using the approach of Tarjan et al. [58] and Hsu et al. [60]. As with other augmentations reported in this thesis, we represent the given k-tree using a minimum vertex separator tree. For a given k-tree, a minimum vertex separator tree is unique and it is different from the standard tree decomposition tree. The following structural characterization for k-trees is presented in [1].

Theorem 7.1.1 *[1] G is a k-tree if and only if*

1. G is connected

2. G has a k-clique but no $k+2$ clique

3. Every minimal vertex separator of G is a k-clique.

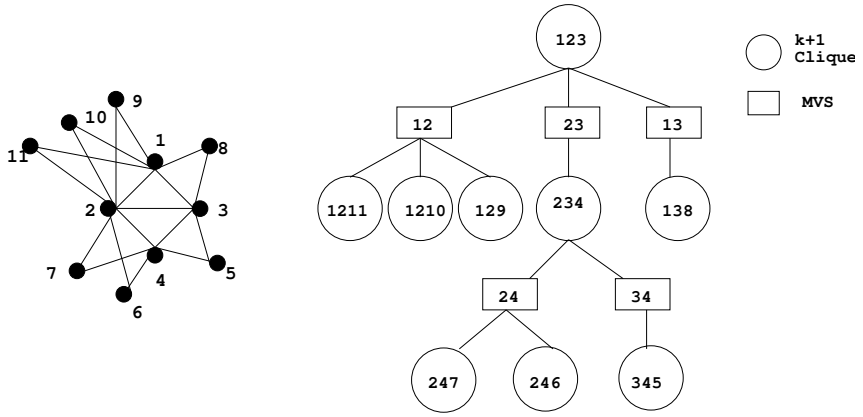


Figure 7.1: A k-tree and its MVS tree

From Theorem 7.1.1, it is easy to see that every minimum vertex separator (MVS) is a k-clique and hence, k-trees are k-connected. Also, every maximal clique is of size $k+1$. For a given k-tree G, we construct a Minimum Vertex Separator tree (MVS-tree) T associated with G as follows: $M(G) = \{S \subset V(G) \mid S \text{ is a minimum vertex separator in } G\}$ and $K(G) = \{K \subset V(G) \mid K \text{ is a } (k+1) \text{ clique in } G\}$. For a vertex $x \in V(T)$, the associated label $label(x)$ is a subset of $V(G)$ such that $label(x) \in M(G)$ or $label(x) \in K(G)$. Let $V_M = \{x \mid label(x) \in M(G)\}$ and $V_K = \{x \mid label(x) \in K(G)\}$. $V(T) = V_M \cup V_K$. For $x, y \in V(T)$, adjacency between x and y is defined as follows: $\{x, y\} \in E(T)$ if $x \in V_M$ and $y \in V_K$ such that $label(x) \subset label(y)$. An example of a MVS-tree is illustrated in Figure 7.1.

7.1.1 A Lower Bound on k-tree Augmentation

Let l denote the number of leaves in T and s denote the number of simplicial vertices in G. By our construction, it is easy to see that $l = s$. The degree of a minimum vertex separator $S \in M(G)$ is the number of $(k+1)$ cliques in G that contain S and it is denoted by $deg(S)$. $deg(S) = |\{K \mid K \in K(G) \wedge S \subset K\}|$. Let $S^{max} = \max_{S \in M(G)} deg(S)$. Let $x \in V_M$ is such that $deg(x) = S^{max}$. Also, $\Delta_k(T) = deg(x)$ such that $deg(x) = S^{max}$.

Lemma 7.1.2 *For any optimum solution $E_{min}(G)$ of G,*
$|E_{min}(G)| \geq \max\{\lceil \frac{s}{2} \rceil, S^{max} - 1\}$.

Proof Given that G is a k-tree, we know that $\kappa(G) = k$. Also, for all simplicial vertices v, $deg(v) = k$. This shows that $\delta(G) = k$. Since for any connected graph H, $\delta(H) \geq \kappa(H)$, it follows that any $(k+1)$-connected graph G' obtained from a k-tree G by augmenting edges is such that $\delta(G') \geq k+1$. This implies that to increase the vertex connectivity of G by one, we need to increase the degree of each simplicial vertex in G by at least one. Therefore, in any vertex connectivity augmentation of G, one must add at least $\lceil \frac{s}{2} \rceil$ edges to G. We now describe another lower bound: we know that in any $(k+1)$-connected graph H, for any two vertices u and v in H, there exists $k+1$ vertex disjoint paths between them. Since S^{max} is a minimum vertex separator of size k, for each u and v in G at most k vertex disjoint paths contain elements from S^{max}. Therefore, to make G a $(k+1)$-connected graph, we must have at least one more path which does not contain elements from S^{max}. Since removal of S^{max} from G creates S^{max} components, to get a path which avoids elements of S^{max} we must to add at least $S^{max} - 1$ edges to G. This proves that $|E_{min}(G)| \geq \max\{\lceil \frac{s}{2} \rceil, S^{max} - 1\}$. Hence the proof. □

For the MVS-tree T, the above lower bound translates into $\max\{\lceil \frac{l}{2} \rceil, \Delta_k(T) - 1\}$ edges.

7.1.2 Augmentation Algorithm

We observe that the MVS-tree is very similar to the biconnected component tree. A node $x \in V_M$ corresponds to an element in B' and $x \in V_K$ corresponds to an element in B. With this observation, we simplify our k-tree augmentation algorithm by calling the subroutine *1-connect-augment* reported in the last chapter with MVS-tree as the input. Let $A = \{\{x, y\} \mid x, y$ are leaves in $T\}$ be the set of edges returned by our algorithm.

Algorithm 8 k-tree connectivity Augmentation: *ktree-augment(Tree T)*

```
/* T is the MVS-tree of a k-tree G */
```
Perform $(k+1)$-connectivity augmentation of G using *1-connect-augment(T)*
For each edge $\{x, y\}$ added into T, add $\{u, v\}$ to G such that $u \in l(x)$ and $v \in l(y)$ and u and v are simplicial vertices in G

Let $E_a(G)$ be the corresponding set of edges in G, $E_a(G) = \{\{u, v\} \mid \{x, y\} \in A$ and $u \in l(x)$ is a simplicial vertex in G and $v \in l(y)$ is a simplicial vertex in $G\}$.

Lemma 7.1.3 *Let G_{aug} be the graph obtained from G by augmenting $E_a(G)$. G_{aug} is $(k+1)$-connected.*

Proof Suppose G_{aug} is not $(k+1)$-connected. This implies that there exists a k size minimum vertex separator S in G_{aug}. Since S is also a separator in G, it follows that S is a k-clique in G. An invariant maintained by our algorithm is that for each minimum vertex separator S', $deg(S') - 1$ edges are added to ensure that the $deg(S')$ components in $G \setminus S'$ are connected in G_{aug}. However, since the removal of S disconnects G_{aug}, it follows that the above invariant is not maintained. This is a contradiction. Hence, there is no k-size vertex separator S in G_{aug}. Therefore, G_{aug} is $(k+1)$-connected. □

Theorem 7.1.4 *Let G be a k-tree. An optimum $(k+1)$-connectivity augmentation of G uses $|E_a(G)| = \max\{\lceil \frac{s}{2} \rceil, S^{max} - 1\}$ edges.*

Proof We know that our algorithm augments $|E_a(G)|$ edges. Therefore, from Lemma 7.1.3, it follows that G augmented with $E_a(G)$ is $(k+1)$-connected. □

Since k-trees are recursive graphs, the construction order [2] of k-trees yields us a minimum vertex separator tree. Moreover, k-trees are chordal and hence, they have perfect elimination ordering [2] of vertices. This shows that MVS-tree can be constructed in $O(n+m)$ time. Also, the number of nodes in the MVS-tree is $O(n)$. Since there is a bijection between a MVS-tree and a biconnected component tree, we know from the previous chapter that an augmenting set can be obtained in $O(n)$ time. Therefore, the overall time complexity to $(k+1)$-connect a k-tree is $O(n+m)$.

7.2 Augmentation in k-connected Chordal Graphs

We now present an algorithm for optimum connectivity augmentation in chordal graphs. Given a k-connected chordal graph, our algorithm augments a minimum set of edges to make it $(k+1)$-connected. A graph is chordal, if there is no induced cycle of length at least 4. Chordal graphs are very important subclass of perfect graphs and chordal graphs are a superclass of k-trees. The highlights of chordal graphs are every minimal

vertex separator is a clique and there exists a perfect elimination ordering of vertices. As with other augmentations reported in this thesis, we represent a k-connected chordal graph using a tree, namely, a *clique separator tree*, the label of whose vertices denote either a minimum vertex separator of size k or a maximal clique of size at least $k+1$ or a $(k+1)$-connected component (a maximal $(k+1)$-connected subgraph) of the graph. From the following observation, it may be noted that such a clique separator tree exists and it is unique.

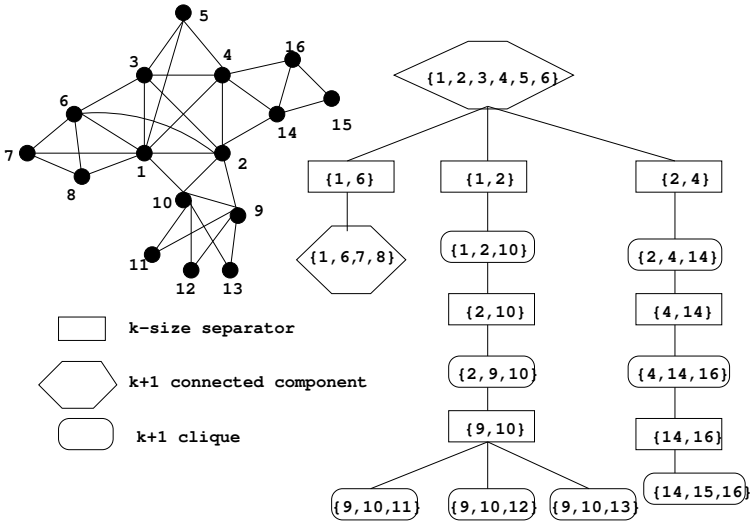


Figure 7.2: A k-connected chordal graph and its clique separator tree

Lemma 7.2.1 *Let G be a k-connected chordal graph and S be a minimum vertex separator in G. Let $\{C_1, \ldots, C_r\}$ denote the set of connected components in $G \setminus S$. Every other minimum vertex separator S' in G is either contained in some C_i or contained in $V(C_i) \cup S$. In other words, there is no S' that spans across the components.*

Proof Suppose there exists S' such that $V(C_i) \cap S' \neq \phi$ and $V(C_j) \cap S' \neq \phi, i \neq j$. Since G is chordal, every minimal vertex separator is a clique. In particular, S' is a clique and this implies that $\{u, v\} \in E(G)$. Hence, there is a path between C_i and C_j. However, we know that such a path does not exist as C_i and C_j are connected components of $G \setminus S$. This shows such S' does not exist. Hence the lemma. □

From the above lemma, observe that the associated tree can be constructed as follows: recursively find a k size vertex separator from the connected components until no longer

possible. Note that in the associated tree T, for $x \in V(T)$, $label(x)$ denotes either a k size vertex separator or a maximal $(k+1)$ clique or a $(k+1)$-connected component of G. For $x, y \in V(T)$, the adjacency between x and y is defined as follows: $\{x, y\} \in E(T)$, if $label(x) \subset label(y)$ and $label(x)$ is a k size vertex separator and $label(y)$ represents either a maximal $(k+1)$ clique or a $(k+1)$-connected component in $G \setminus S$. An example is illustrated in Figure 7.2.

7.2.1 A Lower Bound on Chordal Augmentation

We now analyze the lower bound on the augmentation number and the analysis is very similar to the previous section. Let X denote the set of maximal cliques of size $k+1$ containing *exactly one* k-clique separator in G and the set of $(k+1)$-connected components containing *exactly one* k-clique separator in G. Note that by our construction of T, each element of X corresponds to the *label* of a leaf in T. Clearly, to $(k+1)$-connect G, we must augment at least $\left\lceil \frac{|X|}{2} \right\rceil$ edges and therefore we need at least $\lceil \frac{l}{2} \rceil$ edges, l denotes the number of leaves in T. Another lower bound is obtained due to the k-size vertex separator and is same as the one defined in the previous section. Thus, we conclude that $|E_{min}(G)| \geq \max\{\left\lceil \frac{|X|}{2} \right\rceil, S^{max} - 1\}$. For the clique separator tree T, the above lower bound translates into $\max\{\lceil \frac{l}{2} \rceil, \Delta_k(T) - 1\}$ edges.

7.2.2 Augmentation Algorithm

We observe that the clique separator tree is very similar to the 3-block tree. σ, π, and β vertex of a 3-block tree corresponds to a k-size separator, a maximal $(k+1)$-clique, and a $(k+1)$-connected component, respectively of a clique separator tree. With this observation, we simplify our k-connected chordal graph augmentation algorithm by calling *2-connect-augment* with clique separator tree as the input.

Theorem 7.2.2 *Let G be a k-connected chordal graph. An optimum $(k+1)$-connectivity augmentation of G uses $|E_a(G)| = \max\{\left\lceil \frac{|X|}{2} \right\rceil, S^{max} - 1\}$ edges.*

Proof A proof similar to Theorem 7.1.4 establishes this claim. □

Algorithm 9 Augmentation in k-connected chordal graphs: *k-connected-chordal-augment(Tree T)*

`/* T is the clique separator tree of a k-connected chordal graph G */`
Perform $(k+1)$-connectivity augmentation of G using *2-connect-augment(T)*
For each edge $\{x, y\}$ added into T, add $\{u, v\}$ to G such that $u \in label(x)$ and $v \in label(y)$ and u and v are simplicial vertices in G

The clique separator tree can be obtained from the *clique separator graph* of a chordal graph. The clique separator graph was introduced by Ibaraa in [22]. It is important to note that the clique separator tree is very similar to MVS-tree introduced in the previous section and all combinatorial observations of MVS-tree are true for this tree as well. This shows that if the clique separator tree of a chordal graph is given then a minimum connectivity augmentation in k-connected chordal graphs can be done in linear time. The algorithm in [22] incurs $O(n^3)$ time to compute the clique separator graph of a chordal graph and hence, $O(n^3)$ time to compute a clique separator tree. Therefore, a minimum connectivity augmentation of k-connected chordal graphs can be obtained in $O(n^3)$ time.

Concluding Remarks: Using the framework proposed in the last chapter, we have presented algorithms for connectivity augmentation in k-trees and k-connected chordal graphs. We have proposed a linear-time algorithm for $(k+1)$-connectivity augmentation in k-trees and a cubic time algorithm for $(k+1)$-connectivity augmentation in k-connected chordal graphs. A natural extension of this work is to study connectivity augmentation in graphs of treewidth k. Also, whether connectivity augmentation in general graphs can be done using this framework is open.

CHAPTER 8

Conclusions and Further Research

This thesis shines a light on some of the research questions which came up in the study of the impact of graph operations on vertex connectivity. Our findings include a combinatorial and a complexity-theoretic view. From a combinatorial-view, we have considered the structural characterization of contractible edges in chordal graphs. Also, we have looked at the distribution of contractible edges in this work. Further, we have investigated a structural characterization of graphs in which every minimal vertex separator is a stable set and such a graph has the property that every edge is contractible. Our study of non-edges in 2-connected graphs revealed that cycles are the only 2-connected graphs in which each non-edge is non-contractible. We leave open the following questions for further research on contractible edges.

- Structural characterization of contractible edges in chordality 4 graphs. In general, chordality $c(c \geq 4)$ graphs. Any progress towards characterizing the graph formed by the contractible edges in chordality 4 graphs would be exciting.
- Structural characterization of contractible edges in treewidth k graphs. Algorithmically, whether contractible edges can be used to compute treewidth efficiently seems to be an interesting direction of research.
- Characterization of graphs in which each non-edge is non-contractible.
- We have analyzed the structure of the graph formed by contractible edges by taking connectivity as the parameter of interest. It is curious to look at other graph parameters like treewidth, chordality, girth, etc.

From a complexity-theoretic view, we have investigated two combinatorial optimization problems. The first one is the problem of reducing the (s,t)-vertex connectivity to one using a minimum number of edge contractions. We have shown that this problem is equivalent to the computational problem of finding a minimum connected (s,t)-vertex separator. We have investigated the complexity of connected (s,t)-vertex separator ((s,t)-CVS) and showed that (s,t)-CVS is NP-complete. Further, we have shown that there is no approximation algorithm with approximation ratio $\delta.log^{2-\epsilon}n$, for every $\epsilon > 0$

and for some $\delta > 0$, unless NP has quasi-polynomial Las-Vegas algorithms. We then observed that (s,t)-CVS on chordal bipartite graphs is polynomial-time solvable and NP-complete on graphs with chordality at least 5. From the perspective of parameterized complexity, we have shown that parameterizing above the (s,t)-vertex connectivity is $W[2]$-hard. We leave open the following problems which we came up in the study of (s,t)-CVS.

- Parameterized complexity of (s,t)-CVS parameterized by the (s,t)-vertex connectivity.
- Parameterized complexity of (s,t)-CVS parameterized by the size of (s,t)-CVS.
- It is interesting to look at special graph classes where (s,t)-CVS is polynomial-time solvable.
- We conjecture that cycle is the only graph in which no connected vertex separator exists and in every other graph there exists a connected vertex separator. Any progress towards proving or disproving this conjecture would be exciting.
- Computational complexity of connected vertex separator in general graphs.
- The existence of an approximation algorithm for (s,t)-CVS with approximation ratio $\delta.log^2 n$ for some constant δ, is worth investigating.
- What is the structure of the graph in which every minimal vertex separator is connected?

Our next combinatorial optimization problem is connectivity augmentation which asks for a minimum set of edges to be augmented to a k-vertex connected graph to make it $(k+1)$-vertex connected. Our research contribution includes a unified framework for connectivity augmentation and we have exemplified our framework using four case studies. We have reported biconnectivity augmentation of 1-connected graphs, triconnectivity augmentation of 2-connected graphs, and $(k+1)$-connectivity augmentation of k-trees and k-connected chordal graphs. For further research:

- A natural extension of this work is to study connectivity augmentation in graphs of treewidth k.
- Connectivity augmentation in general graphs using this framework is an interesting direction for further research.
- Complexity of increasing the connectivity by one using minimum number of edge contractions is worth analyzing.

REFERENCES

[1] D.B.West: Introduction to graph theory. Prentice Hall of India (2003)

[2] M.C.Golumbic: Algorithmic graph theory and perfect graphs. Academic Press (1980)

[3] R.G.Downey, M.R.Fellows: Parameterized complexity. Springer Verlag (1999)

[4] R.Niedermeier: Invitation to fixed parameter algorithms. Oxford Lecture Series in Mathematics and its Applications, Oxford University Press (2006)

[5] M.Kriesell: A survey on contractible edges in graphs of a prescribed vertex connectivity. Graphs and Combinatorics, 18, 1-30 (2002)

[6] D.Yang, K.Iwama, K.Naoki: A new probabilistic analysis of Karger's randomized algorithm for min-cut problems. Information Processing Letters, 64(5), 255-261 (1997)

[7] W.T.Tutte: A theory of 3-connected graphs. Indag.Math, 23, 441-455 (1961)

[8] A.Saito, K.Ando, H.Enomoto: Contractible edges in 3-connected graphs. Journal of Combinatorial Theory-B, 42(1), 87-93 (1987)

[9] N.Dean, R.L.Hemminger, B.Toft: On contractible edges in 3-connected graphs. Congressus Numerantium, 58, 291-293 (1987)

[10] N.Dean, R.L.Hemminger, K.Ota: Longest cycles in 3-connected graphs contain three contractible edges. Journal of Graph Theory, 13(1), 17-21 (1989)

[11] M.N.Ellingham, R.L.Hemminger, K.E.Johnson: Contractible edges in longest cycles in non-hamiltonian graphs. Discrete Mathematics, 133, 89-98 (1994)

[12] K.Fujita: Maximum number of contractible edges on hamiltonian cycles of a 3-connected graph. Graphs and Combinatorics, 18(3), 447-478 (2005)

[13] N.Martinov: A recursive characterization of 4-connected graphs. Discrete Mathematics, 84, 105-108 (1990)

[14] N.Martinov: Uncontractible 4-connected graphs. Journal of Graph Theory, 6, 343-344 (1982)

[15] K.Ando, K.Kaneko, A.Kawarabayashi: Vertices of degree 5 in a 5-contraction critical graph. Graphs and Combinatorics, 21(1), 27-37 (2005)

[16] C.Thomassen: Non-separating cycles in K-connected graphs. Journal of Graph Theory, 5, 351-354 (1981)

[17] D.Haglin, W.McCuaig, S.Venkatesan: Contractible edges in 4-connected maximal planar graphs. Ars Combinatorica, 31, 199-203 (1991)

[18] N.Dean: Distribution of contractible edges in k-connected graphs. Journal of Combinatorial Theory Series B, 48, 1-5 (2003)

[19] M.Kriesell: Contractible non-edges in triangle-free graphs. Graphs and Combinatorics, 15, 429-439 (1999)

[20] M.Kriesell: Contractible non-edges in 3-connected graphs. Journal of Combinatorial Theory Series B, 74, 192-201 (1998)

[21] Y.Shibata: On the tree representation of chordal graphs. Journal of Graph Theory, 12(3), 421-428 (1988)

[22] L.Ibarra: Clique separator graph for chordal graphs. Discrete Applied Mathematics, 157 (8), 1737-1749 (2009)

[23] G.A.Dirac: On rigid circuits. Abhandlungen aus dem Mathematischen Seminar der Universitat(German), Hamburg, 25,71-76 (1961)

[24] S.H.Whitesides: An algorithm for finding clique cutsets. Information Processing Letters, 12, 31-32 (1981)

[25] A.Brandstadt, F.F.Dragan, V.B.Le, T.Szymczak: On stable cutsets in graphs. Discrete Applied Mathematics, 105, 39-50 (2000)

[26] D.Marx, B.O'Sullivan, I.Razgon: Treewidth reduction for constrained separation and bipartization problems. In Proceedings of 27th International Symposium on Theoretical Aspects of Computer Science (STACS), pp. 561-572 (2010)

[27] N.Trotignon, K.Vuskovic: A structure theorem for graphs with no cycle with a unique chord and its consequences. Journal of Graph Theory, 63, 31-67 (2010)

[28] M.R.Garey, D.S.Johnson: Computers and intractability: A guide to the theory of NP-completeness. W.H.Freeman and Company (1979)

[29] O.Oellermann, J.P.Spinrad: A polynomial time algorithm for testing whether a graph is 3-steiner distance hereditary. Information Processing Letters, 55, 149-154 (1995)

[30] R.Diestel: Graph theory. Electronic Edition, Springer-Verlog (2005)

[31] T.H.Cormen, C.E.Leiserson, R.L.Rivest, C.Stein: Introduction to algorithms. McGraw-Hill (2002)

[32] W.H.Cunningham: The optimal multiterminal cut problem. Reliability of Computer and Communication Networks, 5, 105-120 (1989)

[33] R.J.Lipton, R.E.Tarjan: A separator theorem for planar graphs. SIAM Journal of Applied Mathematics, 36(2), 177-189 (1979)

[34] T.N.Bui, C.Jones: Finding good approximate partitions is NP-hard. Information Processing Letters, 42, 153-159 (1992)

[35] D.Marx: Parameterized graph separation problems. Theoretical Computer Science, 351, 394-406 (2006)

[36] R.Downey, V.Estivill-Castro, M.Fellows, E.Prieto, F.Rosamond: Cutting up is hard to do: The parameterized complexity of k-cut and related problems. Electronic Notes in Theoretical Computer Science, 78, 1-15 (2003)

[37] J.Chen, Y.Liu, S.Lu: An improved parameterized algorithm for the minimum node multiway cut problem. Algorithmica, 55, 1-13 (2009)

[38] N.Garg, V.V.Vazirani, M.Yannakakis: Multiway cuts in node weighted graphs. Journal of Algorithms, 50, 49-61 (2004)

[39] T.F.Gonzalez: On the computational complexity of clustering and related problems. In proceedings of 10th IFIP Conference on System Modeling and Optimization, pp.174-182 (1981)

[40] J.A.Hartigan: Clustering algorithms. Wiley, New York (1975)

[41] D.S.Johnson, C.R.Aragon, L.A.McGeoch, C.Schevon: Optimization by simulated annealing: an experimental evaluation of graph partitioning. Operations Research, 37, 865-892 (1989)

[42] L.Tao, Y.C.Zhao, K.Thulasiraman, M.N.S.Swamy: Simulated annealing and tabu search algorithms for multiway graph partitioning. Journal of Circuits, Systems and Computers, 2, 159-185 (1992)

[43] S.Even: An algorithm for determining whether the connectivity of a graph is at least k. SIAM Journal of Computing, 4(4), 393-396 (1975)

[44] S.Even, R.E.Tarjan: Network flow and testing graph connectivity. SIAM Journal of Computing, 4(4), 507-518 (1975)

[45] H.Gabow: Using expanders to find vertex connectivity. Journal of Association for Computing Machinery, 53(5), 800-844 (2006)

[46] A.Kanevsky: Finding all minimum size separating vertex sets in graphs. Networks, 23, 533-541 (1993)

[47] T.Kloks, D.Kratsch: Listing all minimal separators of a graph. SIAM Journal of Computing, 27(3), 605-613 (1998)

[48] M.Mahajan, V.Raman: Parameterizing above guaranteed values: Maxsat and maxcut. Journal of Algorithms, 31(2), 335-354 (1999)

[49] K.White, M.Farber, W.R.Pulleyblank: Steiner trees, connected domination and strongly chordal graphs. Networks, 15, 109-124 (1985)

[50] U.Feige: A threshold of $\ln(n)$ for approximating set cover. Journal of Association for Computer Mathematics 45, 634-652 (1998)

[51] E.Halperin, R.Krauthgamer: Polylogarithmic inapproximability. Proceedings of 35th annual ACM Symposium on Theory of Computing (STOC) (2003)

[52] T.Kloks, H.Bodlaender, H.Muller, D.Kratsch: Computing treewidth and minimum fill-in: All you need are the minimal separators. Proceedings of ESA'93, LNCS 726, 260-271 (1993)

[53] M.Cesati: The turing way to parameterized complexity. Journal of Computer System Sciences, 67, 654-685 (2003)

[54] S.Dreyfus, R.Wagner: The steiner problem in graphs. Networks, 1, 195-207 (1971)

[55] B.Courcelle: The monadic second order logic of graphs: Recognizable sets of finite graphs. Information and Computation, 85, 12-75 (1990)

[56] G.Gottlob, S.T.Lee: A logical approach to multicut problems. Information Processing Letter, 103, 136-141 (2007)

[57] L.A.Vegh: Augmenting undirected node connectivity by one. In Proceedings of 42nd ACM Symposium on Theory of Computing (STOC), pp. 563-572 (2010)

[58] K.P.Eswaran, R.E.Tarjan: Augmentation problems. SIAM Journal of Computing, 5, 653-665 (1976)

[59] A.Rosenthal, A.Goldner: Smallest augmentation to biconnect a graph. SIAM Journal of Computing, 6, 55-66 (1977)

[60] T.S.Hsu, V.Ramachandran: On finding a smallest augmentation to biconnect a graph. SIAM Journal of computing, 22, 889-912 (1993)

[61] T.S.Hsu: Simpler and faster biconnectivity augmentation. Journal of Algorithms, 45, 55-71 (2002)

[62] T.Watanabe, A.Nakamura: 3-connectivity augmentation problems. In Proceedings of IEEE International Symposium on Circuits and Systems, pp. 1847-1850 (1988)

[63] T.S.Hsu, V.Ramachandran: A linear time algorithm for triconnectivity augmentation. In proceedings of 32nd Annual IEEE symposium on Foundations of computer science(FOCS), pp.548-559 (1991)

[64] T.S.Hsu: On four connecting a triconnected graph. Journal of Algorithms, 35, 202-234 (2000)

[65] B.Jackson, T.Jordan: Independence free graphs and vertex connectivity augmentation. Journal of Combinatorial Theory Series B, 94, 31-77 (2005)

[66] T.Jordan: Two NP-complete augmentation problems. Technical Report, Department of Mathematics and Computer Science, Odense University, Denmark (1997)

[67] T.Watanabe, A.Nakamura: Edge-connectivity augmentation problems. Journal of Computer System Sciences, 35(1), 96-144 (1987)

[68] G.R.Cai, Y.G.Sun: The minimum augmentation of any graph to k-edge connected graph. Networks, 19, 151-172 (1989)

[69] W.Mader: A reduction method for edge-connectivity in graphs. Annals in Discrete Mathematics, 3, 145-164 (1978)

[70] A.Frank: Augmenting graphs to meet edge-connectivity requirements. SIAM Journal of Discrete Mathematics, 5(1), 22-53 (1992)

[71] A.Frank, T.Jordan: Minimal edge-coverings of pairs of sets. Journal of Combinatorial Theory Series B, 65, 73-110 (1995)

[72] A.Frank: Connectivity augmentation problems in network design. Mathematical Programming, 66, 34-63 (1994)

[73] K.Steiglitz, P.Weiner, D.J.Klietman: The design of minimum-cost survivable networks. IEEE transactions on Circuit Theory, CT-16, 455-460 (1969)

[74] H.Frank, W.Chou: Connectivity considerations in the design of survivable networks. IEEE transactions on Circuit Theory, CT-17, 486-490 (1970)

[75] S.P.Jain, K.Gopal: On network augmentation. IEEE Transactions on Reliability, R-35, 541-543 (1986)

[76] R.Tarjan: Depth first search and linear graph algorithms. SIAM Journal of Computing, 1(2), 146-160 (1972)

[77] W.T.Tutte: Connectivity in graphs. University of Toronto Press (1966)

[78] J.E.Hopcroft, R.E.Tarjan: Dividing a graph into triconnected components. SIAM Journal of Computing, 2, 135-158 (1973)

[79] T.S.Hsu, V.Ramachandran: A linear time algorithm for triconnectivity augmentation. In proceedings of IEEE symposium on Foundations of Computer Science(FOCS), pp.548-559 (1991)

LIST OF PAPERS BASED ON THESIS

Papers in Refereed Journals

1. "On the Structure of Contractible Edges in k-connected Partial k-trees", N.S.Narayanaswamy, N.Sadagopan and L.Sunil Chandran, Graphs and Combinatorics 25, 1-13, (2009) Springer Publications.

2. "On the Structure of Contractible Vertex Pairs in Chordal Graphs", N.S.Narayanaswamy and N.Sadagopan, Electronic Notes in Discrete Mathematics, 33, 29-36, (2009). Elsevier Publications.

3. "Non-contractible non-edges in 2-connected graphs", Anita Das, Mathew C. Francis, Rogers Mathew, and N. Sadagopan, Information Processing Letters, 110 (23), 1044-1048, (2010). Elsevier Publications.

4. "A Unified Framework for Bi(Tri)connectivity and Chordal Augmentation", N.S.Narayanaswamy and N.Sadagopan, **accepted** at International Journal of Foundations of Computer Science.

Papers in International Conferences

1. "Contractible Non-Edges in Chordal Graphs", N.S.Narayanaswamy and N.Sadagopan, In Proceedings of International Conference on Graph Theory and Applications(ICGTA), pp. 222-233, (2008).

2. "Connectivity Augmentation in Chordal Graphs Preserving Chordality", N.S.Narayanaswamy and N.Sadagopan, In Proceedings of 16^{th} International Conference on Advanced Computing and Communications(ADCOM), pp. 11-16, (2008).

3. "A Novel Data Structure for Biconnectivity, Triconnectivity, and k-tree Augmentation", N.S.Narayanaswamy and N.Sadagopan, In Proc. of 17^{th} Computing: The Australasian Theory Symposium (CATS), CRPIT Series, 119, 45-54, (2011).

4. "A Characterization of all Stable Minimal Separator Graphs", Mrinal Kumar, Gaurav Maheswari, and N.Sadagopan, accepted for poster presentation in European Conference on Combinatorics, Graph Theory and Applications (EUROCOMB'11).

Under Journal Review

1. "Complexity of Connected (s, t)-Vertex Separator", N.S.Narayanaswamy and N.Sadagopan.

2. "A Note on Stable Minimal Vertex Separator Graphs", Mrinal Kumar, Gaurav Maheswari, and N.Sadagopan.