Utilizing Graph Classes for Community Detection in Social and Complex Networks

by

James Nastos

B.Math., University of Waterloo, 2000
B.Ed., University of British Columbia, 2002
M.Sc., University of Alberta, 2006

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

THE COLLEGE OF GRADUATE STUDIES

(Interdisciplinary Studies - Optimization)

THE UNIVERSITY OF BRITISH COLUMBIA

(Okanagan)

April 2015

Abstract

Social network analysis is a cross-disciplinary study of interest to mathematicians, physicists, computer scientists and sociologists. It deals with looking at large networks of interactions and extracting useful or meaningful information from them. One attribute of interest is that of identifying social communities within a network: how such a substructure should be defined is a widely-studied problem in itself. With each new definition, there is a need to study in what applications or context such a definition is appropriate, and develop algorithms and complexity results for the computation of these clusterings.

This thesis studies problems related to graph clustering, motivated by the social networking problem of community detection. One main contribution of this thesis is a new definition of a specific kind of family-like community, accompanied by theoretical and computational justifications. Additional results in this thesis include proofs of hardness for the quasi-threshold editing problem and the diameter augmentation problem, as well as improved exact algorithms for cograph and quasi-threshold edge deletion and vertex deletion problems.

Preface

Many of the results contained in this thesis have already appeared in published print.

- The results in Section 2.3 are joint work with my supervisors Y. Gao and D. R. Hare and appear in the Discrete Mathematics paper: “The cluster deletion problem for cographs” [56].

- Sections 2.4, 2.5 and 2.6 appear in the Social Networks paper: “Familial groups in social networks” [110].

- The diameter-related results in Section 4.3 have been published in the Discrete Applied Mathematics paper [57], although we present an alternate proof of the diameter-2 theorem in this thesis, originally printed in an earlier arxiv manuscript [107].

- The work shown in sections 4.4.4 and 4.4.5 is my work done as part of a project originating from Ajay Sridharan’s MSc thesis (UVictoria) [132] and appear in the publication [133].

- The results in Chapter 5 were preliminarily published in the LNCS conference proceedings of the 2010 Conference on Combinatorial Optimization and Algorithms (COCOA) [108] and later in journal form in [109].

Table of Contents

List of Tables

List of Figures

Acknowledgements

Firstly, I would like to thank Unit 5 of UBCO for their years of hospitality, support, and opportunity. An interdisciplinary PhD has proven to be a fruitful endeavor and I appreciate the effort put towards creating and evolving such a graduate program. I also greatly appreciate the opportunities I had to teach undergraduates, especially the lecturing appointments I had in 2011 and 2012.

I also want to thank everyone in the department I have had dealings with, from fellow graduate student officemates to professors who I marked exams with to the departmental support staff. I feel the relationships I made here were collegial yet friendly and personal, and have contributed to an amazing experience for me.

In addition to departmental financial support, I would like to especially thank Dr. Y. Gao and for his financial support throughout my PhD.

I would also like to thank Dr. Y. Gao for his mentorship and encouragement over the duration of this PhD. His ability to identify the strengths of his students and how they can contribute to a research project has been the driving force behind all the work in this thesis.

And I would like to thank both my supervisors, Dr. Y. Gao and Dr. D. R. Hare for their collaboration with and supervision of all the work in this thesis. They each brought a style of problem solving to our collaboration which resulted in successes I could not have achieved on my own. I also must thank them for their infinite patience.

And finally, I would like to thank my family for their love, support and hugs.

Dedication

Dedicated to my father, Athanasios (Tommy) Nastos, who offered me (among so many other things) every academic opportunity I ever wanted.

Chapter 1

Introduction

The field of graph theory has generally been application-driven, even from its earliest days. Indeed, the first theorem on graph theory from 1736 was motivated by a path-finding problem over bridges [44] and its associated algorithms have recently been used in DNA fragment assembly [119]. While graph theory has found its place in pure mathematics, the branch of algorithmic graph theory has flourished alongside the development and improvement of computer technology.

Social networks are graph structures that had been studied by sociologists long before the existence of online social networks. The popularity and accessibility of online social networks in recent years has brought the notion of graph networks to the forefront of much research in sociology, physics, marketing, computer science, mathematics, epidemiology of infectious diseases, and more. The analysis of the structure of social networks has spread beyond that of an academic interest as companies (such as Klout) that specialize in social media analytics are noticeably gaining popularity.

Much of this thesis is inspired by the phenomenon of *clustering* in networks. In social networks, clustering can take the form of *communities*, *hierarchical organization* or *structural equivalence*, as examples.

The rest of Chapter 1 reviews background material on algorithms, complexity theory, graph classes and graph modification problems.

Chapter 2 introduces and explores a new definition for modeling social communities as *familial groups* through the graph-theoretic concept of *quasi-threshold graphs*. Computational problems for finding familial groups are given, including a problem reduction and exact algorithms. Some computational results are also included to strengthen the case for a social interpretation of familial groups.

Chapter 3 offers a formal framework for potentially extending the idea of familial groups to directed networks. Chapter 4 discusses the well-studied phenomenon of *power-law* distributions observed in social network measures such as vertex degree and community size. An equally popular phenomenon, the *small-word* property, is discussed and a study on the computational complexity of the relevant measure of *graph diameter* is given. Chapter 5 presents the main algorithms of the thesis for various graph modification problems. We summarize the results and mention some open problems and future directions in Chapter 6.

1.1 Definitions

We begin with a set of necessary definitions of terms that will be used throughout the thesis.

1.1.1 Graphs and Networks

Our graph-theoretical definitions follow the reference book *Graph Classes: A Survey*[16], and we refer the reader to that book for any definitions or basic concepts not included in this thesis.

A *graph* is a structure composed of a set of objects or *vertices* and a set of *edges*, each edge joining two vertices. More formally, a graph is a pair (V, E) where V is a set and E is a set of unordered pairs of V. Unless otherwise noted in the discussion, a graph will be understood to be simple (no duplicate edges, no edge joining a vertex to itself). Vertices are also sometimes referred to as *nodes*.

An edge e that joins two vertices u and v will be written as $e = uv$ or equivalently, $e = vu$, $e = \{u, v\}$. When $e = uv$ is an edge of a graph, we will say that *e is uv*, or that *e is incident on u* and v or that *u is adjacent to v*. We also say that the *endpoints* of e are u and v. The *degree* of a vertex v is the number of distinct edges having v as an endpoint.

The vertex set V of a graph $G = (V, E)$ can be written as $V(G)$ or V_G, and similarly the edge set E of G will be written as $E(G)$ or E_G. Unless otherwise defined, a convention we adopt here is that when there is only one graph in consideration, the symbols n and m are reserved for $n = |V|$ and $m = |E|$. Additional notation will be defined in cases where multiple graphs are involved.

When edges are ordered pairs, E can be described as a subset of $V \times V$. Such edges are called *arcs* or *directed edges*, and a graph involving directed edges will be called a *digraph.*

A *subgraph* $H = (V_H, E_H)$ of a graph $G = (V_G, E_G)$ is another graph where $V_H \subseteq V_G$ and $E_H \subseteq E_G$. An important type of subgraph that will be used throughout this thesis is an *induced subgraph*, where if u and v are vertices in H, there is an edge $uv \in E_H$ if and only if $uv \in E_G$. Given a graph G, we may specify a subset W of the vertex set $V(G)$ and speak of the induced subgraph on W, which is the unique graph having vertex set

W and every edge of $E(G)$ which joins two vertices of W. The *size* of an induced subgraph H is the number of vertices in H.

The *complement* $\overline{G} = (V, \overline{E})$ of a graph $G = (V, E)$ is another graph with the same set of vertices, but where $uv \in \overline{E}$ if and only if $uv \notin E$. A set of vertices which are pairwise adjacent is called a *clique* and a clique is called a *maximal clique* if it is not contained as a subgraph in any other clique. A *maximum clique* of a graph is a clique with the largest number of vertices. A maximum clique is a maximal clique, but not every maximal clique is a maximum clique. A clique of size k will be called a k-clique. The complement of a clique is a set of vertices with no edges, and it is called a *stable set* or an *independent set.*

A *colouring* of a graph is an assignment of a label to each vertex of the graph such that no two adjacent vertices are given the same label. These labels will be called *colours.* When the colouring uses k colours to colour the n vertices in this manner, it is called a k-colouring. The chromatic number $\chi(G)$ of a graph is the smallest number k for which G has a k-colouring. Another way to describe this is that a graph is k-coloured if we partition its vertices into k disjoint sets, each set inducing an independent set. Note that if a graph contains a k-clique, then $\chi(G) \geq k$.

A sequence $v_1, v_2, \ldots, v_k$ of pairwise distinct vertices is a *path* of a graph G if $v_i v_{i+1} \in E(G)$. The *length* of such a path is $k - 1$. A path is also called a *cycle* if $v_1 v_k \in E(G)$. The *length* of such a cycle is k.

If $v_1, v_2, \ldots, v_k$ is a path (respectively, cycle) of a graph G, we say that an edge $e \in E(G)$ that joins two vertices of the path (cycle) is a *chord* of the path (cycle) if e is not an edge of the path (cycle). A path (resp. cycle) is *chordless* if it contains no chords. The *endpoints* of a path $v_1, v_2, \ldots, v_k$ are v_1 and v_k.

In this thesis, we denote the chordless paths and chordless cycles on k vertices as P_k and C_k, respectively. A P_2 is simply an edge and a C_3 is a clique on 3 vertices, also called a *triangle.*

Two graphs G and H are called *isomorphic* if there is a bijective function f from $V(G)$ to $V(H)$ such that $uv \in E(G)$ if and only if $f(u)f(v) \in E(H)$.

When a subgraph H on k vertices is isomorphic to a path (resp. cycle), we will say that H *is* a P_k (C_k). When the induced subgraph on a set of vertices is a P_k (C_k), we may refer to that vertex set itself as being a P_k (C_k).

Two vertices are called *connected* if there is some path having those two vertices as endpoints. A *component* (or, equivalently, a *connected component*), C of a graph G is a maximal subgraph such that every two vertices in C are connected. We write $C \subseteq G$ to denote that C is a subgraph of G.

A connected graph which has no cycle is called a *tree*. A graph in which every connected component is a tree is called a *forest*.

1.1.2 Complexity Theory

Algorithm Analysis

A complete treatment of the analysis of algorithms can be found in a standard textbook such as [30], and we refer the reader to that text for any additional definitions and basic concepts not mentioned here. Our default assumption in analysing algorithms in this thesis is that an item of the input data can be accessed in constant time. This is realistic in practice provided the data in use is completely stored in RAM (*random access memory*). Any exceptions to this assumption will be explicitly stated.

The following notation is standard in the analysis of function growth rates:

> function $f(n)$ is $O(g(n))$ if there exists a positive constant c such that $f(n) < c * g(n)$ for all sufficiently large n.
>
> function $f(n)$ is $\Omega(g(n))$ if there exists a positive constant c such that $f(n) > c * g(n)$ for all sufficiently large n.
>
> function $f(n)$ is $\Theta(g(n))$ if $f(n)$ is both $O(g(n))$ and $\Omega(g(n))$.

As is standard, algorithm runtime or space-complexity will be analysed by worst-case analysis throughout this thesis unless otherwise stated. Growth rates for runtime or space will be expressed in terms of input size and/or input parameters. For a graph with n vertices and m edges, an algorithm is considered to run in *linear time* if it runs in $O(m + n)$-time. A *polynomial time* algorithm is an algorithm whose runtime can be bounded by some polynomial function of the input size.

An *exponential time* algorithm is an algorithm whose (worst case) runtime is $\Omega(c^n)$ where n is an input parameter and $c > 1$ is a constant. When multiple input parameters such as n and m are given and an algorithm runs in $O(2^m p(n))$ where $p(n)$ is some polynomial function, we may say this algorithm is *exponential in* m and also *polynomial in* n*)*.

Reductions

A problem P is called *polytime solvable* or simply *polytime* if there is a polynomial-time algorithm solving every instance of P. Unless P is stated to be an optimization problem, P is understood to be a decision problem where every instance I of P is either a *yes*- or *no*-instance. The task of a decision problem is to determine whether a given instance is a yes-instance or no-instance.

A decision problem P_1 can be shown to be at least as hard as a problem P_2 if one can show that every instance I_2 of P_2 reduces to a problem instance I_1 of P_1 in such a way that I_2 is a yes-instance of P_2 if and only if I_1 is a yes-instance of P_1. With such a reduction, any algorithm that solves problem P_1 can be used to solve any instance of P_2 by reducing the instance of P_2 to an instance of P_1 and using the algorithm that solves P_1 to find the correct answer.

Let R be a reduction that transforms instances of problem P_2 to an instance of problem P_1. If P_1 can be solved with a polynomial time algorithm and the reduction can be computed in polynomial time, then composing the reduction with the algorithm as described earlier will yield a polytime solution for P_2. If, on the other hand, there is no polytime solution to P_2, yet P_2 polynomially reduces to P_1, then there cannot be a polytime algorithm solving P_1. This concepts allows for a definition of computational hardness: that if P_2 is sufficiently hard, then P_1 is at least as hard.

When a proposed solution to an instance of a decision problem is given to affirm that this instance is a yes-instance (such as a set of vertices for Clique or a truth assignment for 3-Sat) and this solution can be verified within polynomial time of the problem size, then the problem is said to be in NP.

A problem P_1 is *NP-hard* if every problem in NP polynomially reduces to P_1. If an NP-hard problem is also in NP itself, then it is called *NP-complete*.

The Cook-Levin Theorem [28] showed that the Boolean Satisfiability Problem is NP-complete, and Karp's consequences [82] used polytime reductions to show that many other problems such as Clique, Independent Set, 3-Sat are also NP-complete.

To date, there is no proof that an NP-complete problem cannot be solved in polynomial time. Since a polynomial time solution to any one of the thousands of known NP-complete problems would imply a polytime solution to every NP-complete problem, and that no polytime algorithm has been found to solve an NP-complete problem, it is generally accepted that the class of NP-complete problems form a set of computationally-hard problems. This notion of computational hardness has allowed researchers to identify a problem as NP-hard and then attempt to deal with solving these problems with specific algorithm-design strategies, such as those from *parameterized complexity* (next subsection), approximation algorithms, heuristics or other search strategies.

Parameterized Complexity

Given a problem P and an algorithm solving it, the algorithm is typically measured in terms of the input size n of P. In the case that the input to a problem is a graph, the runtime is usually written in terms of $n = |V|$ and $m = |E|$. Sometimes, in addition to a problem instance, the input may also accept a parameter k, in which case we say that the problem is *parameterized*.

A problem which accepts a parameter k is called *fixed-parameter tractable* (or FPT) if there exists an algorithm solving the problem with runtime $O(f(k)n^c)$ where n is the input size, f is a function of k which does not depend on n and c is a constant. When the value k is fixed, this is essentially a polynomial runtime, and in particular for any fixed k it is the same polynomial (up to coefficients). FPT algorithms have received much attention lately as many NP-hard problems have been shown to be fixed-parameter tractable. For instance, the $\textsc{Vertex Cover}(G, k)$ problem is a well-known NP-complete problem but can be solved in $O(2^k n)$-time by a simple search tree method, selecting one of two possible endvertices of an uncovered edge. Note that this runtime is linear in n for any fixed k. Analogous to the idea of NP-hardness, there is a measure of hardness for parameterized problems which depends on parameterized reductions.

Parameterized problems are classified into a hierarchy of problem classes: FPT, $W[1]$[1], $W[2], \ldots, W[t], \ldots$. The *weighted* $3\textsc{Sat}(k)$ parameterized problem asks whether an instance of $3\textsc{Sat}$ has a satisfying assignment with Hamming weight equal to k. Weighted $3\textsc{Sat}(k)$ and $\textsc{Clique}(G, k)$ are representative problems in $W[1]$. A parameterized reduction is a Turing reduction taking time $O(f(k)p(n))$ where $p(n)$ is a polynomial in the input size and $f(k)$ is an arbitrary function which does not depend on n. Additionally, an instance (x, k) reducing to an instance (x', k') must produce a parameter k' dependent only on k and not on $n = |x|$. The class of all fixed-parameter tractable problems, FPT, is contained in $W[1]$. The complete problems for $W[1]$ are not expected to be in FPT, as it has been shown that

[1] The class $W[i]$ is defined as the class of parameterized problems that can reduce to a Weighted Circuit Satisfiability problem for weft-i circuits. Elaborating on this definition diverts us away from our discussion of graph problems, and the reader is only required to know that there are representative graph problems for $W[1]$ and $W[2]$ for the purposes of this thesis. Some representative problems for these classes are given in the following paragraph.

if FPT = $W[1]$ then all problems in NP can be solved in $O(2^{o(n)})$-time [41].

Weighted 3Sat(k) and Clique(G,k) are $W[1]$-complete. Dominating Set(G,k) is $W[2]$-complete. Discussion of these results and a thorough introduction to parameterized problems can be found in [115]. Being parameterized-hard also has implications for the approximatibility of the problem. For instance, a problem which is $W[1]$-hard does not have an efficient polynomial-time approximation scheme (EPTAS) unless $W[1] =$ FPT [103].

1.2 Computational Problems on Graphs

1.2.1 Max Clique

Finding a maximum clique is fundamental to many network clustering methods. The corresponding decision problem is formulated as follows:

Problem 1. Clique(G,k)
Input: A graph $G = (V,E)$ and a positive integer k.
Task: To determine if there exists a set $S \subseteq V$ of size at least k such that for every $u,v \in S$, uv is an edge.

This problem is NP-complete [82] and $W[1]$-complete [115].

Since an independent set is the complement of a clique, the problem of finding a maximum independent set is computationally equivalent to the maximum clique problem on the graph complement.

1.2.2 Dominating Set

A *dominating set* in a graph $G = (V,E)$ is a set of vertices $S \subseteq V$ such that every vertex of G is either in S or adjacent to some vertex of S. Dominating sets and their variants have appeared in the context of many applications, see for example [138] or [137].

Problem 2. Dominating Set(G,k)
Input: A graph $G = (V,E)$ and a positive integer k.
Task: To determine if there exists a set $S \subseteq V$ of size at most k such that for every $v \in V \setminus S$ there is some $s \in S$ where sv is an edge.

This problem is NP-complete and $W[2]$-complete [115].

1.2.3 Diameter Augmentation

For any two vertices x,y in a graph, a (x,y)-path is a path having endpoints x and y. Let $dist(x,y)$ be the *distance* between x and y, defined as the number of edges in a shortest path joining them, if one exists. If such a path does not exist, we may define $dist(x, y)$ to be ∞ where every real number r has the property that $r < \infty$, but we do not explicitly need this special case in this thesis. When intermediate vertices and edges of a (x, y)-path are restricted to a particular subgraph H of G, the distance between x and y will be denoted $dist_H(x, y)$. A graph G has *diameter t* if $\max_{x,y} dist(x, y) = t$, and we write $diam(G) = t$.

Graph diameter has been studied for its combinatorial structure as well as for the wide-ranging applications. A decision problem formulation is:

Problem 3. Diameter-t Augmentation(G, k)
Input: A graph $G = (V, E)$ and a positive integer k.
Task: To determine if there exists a set S of at most k edges such that resulting graph $(V, E \cup S)$ has diameter t.

This problem was first shown NP-hard for $t = 3$ in [128] and was later shown to remain hard for the $t = 2$ case [95]. Note that the case of $t = 1$ is trivially polynomial-time solvable as adding an edge between every pair of nonadjacent vertices is necessary.

We prove in this thesis that Diameter-2 Augmentation is $W[2]$-hard in Section 4.3. We show that Diameter-t Augmentation is $W[2]$-hard for $t > 2$ in [57].

1.2.4 Graph Modification Problems

An important class of problems studied in this thesis can be described as *graph modification problems*. The general form of a graph modification problem takes a graph and asks if it can be altered using at most k operations so that the resulting graph has a desired property.

An example of a graph modification problem is the Feedback Vertex Set problem, which is one of the earliest-known NP-complete problems [82]. A *feedback vertex set* is a set of vertices whose removal from a graph results in an acyclic graph. The decision problem asks if there is a feedback vertex set of size at most k.

Many of the above problems can also be stated as graph modification problems: for example, finding a maximum clique is equivalent to finding the minimum number of vertices that can be removed to leave behind a clique.

An *edge edit* is the operation of either adding or deleting an edge. A large class of graph modification problems revolve around edge edits, asking if there is a set of at most k edge edits in order to turn a given graph into one of a type-$\mathcal{C}$, where $\mathcal{C}$ is some set of graphs having a desired property. This edge editing problem is usually called the *$\mathcal{C}$-editing problem.* When restricting the edge edits to edge *deletions*, the corresponding problem is usually referred to as a *$\mathcal{C}$-deletion problem.* Similarly, the *$\mathcal{C}$-addition* or *$\mathcal{C}$-completion problem* only adds edges in order to obtain a graph of type $\mathcal{C}$.

A conceptually simple way to extract the dense clusters of a network is to modify a given graph with the fewest number of edge edits in order to leave behind a collection of disjoint cliques. This is precisely the *cluster editing* problem (definitions and discussion of cluster graphs appear in Section 1.3.1). Cluster editing and deletion has been studied extensively in the scope of FPT algorithms, and discussion of this will be revisited a number of times in this thesis.

Yannakakis shows that vertex-deletion problems to many types of target structures are NP-hard [148]. Elmallah and Colbourn give hardness results for many edge-deletion problems [43].

When the graph property of belonging to a class $\mathcal{C}$ can be characterized by a finite list of forbidden induced subgraphs, it was shown by Cai [19] that the graph modification problem allowing up to k edge and/or vertex edits is fixed-parameter tractable.

As we define several graph classes in the next section, we will not only discuss their defining properties and characteristics but we will also discuss the current state of their associated graph modification problems.

1.3 Graph Classes

A central concept that is used throughout this thesis involves special *graph classes* which are sets of graphs defined with a particular structure. The study of graph classes comes from the fact that modeling certain applications on graphs often implies an inherent structure in the resulting graph which can be exploited.

For example, if we are given the task of assigning rooms to n seminars, each of which has a specified start time and end time, then we can model this as a graph colouring problem by the set of all seminars being the vertex set and two seminars are adjacent if they have an overlap in their time interval (a time conflict). Then a colouring of these vertices corresponds to a room assignment of the seminars, each colour class having no edges (no time conflicts) within it. A minimum colouring corresponds to a solution using the fewest number of rooms.

Karp's consequences [82] showed that finding the chromatic number is

NP-complete (in fact, it showed that simply determining whether a given graph can be coloured with 3 colours is itself an NP-complete problem). But our room-assignment problem can be solved in polytime due to the fact that the resulting graph obtained from intersections of intervals on a time line has a special structure. Note, for example, that it is impossible for graph obtained in this way to contain a chordless cycle of size 4 or more. The resulting graphs are called *interval graphs* and it can be shown that a greedy method can optimally colour an interval graph in linear time.

Let H be some fixed graph. If a graph G does not contain an induced subgraph which is isomorphic to H, then G is called *H-free*. The discussion above states that if a graph G is an interval graph, then G is C_k-free for each $k \geq 4$. There are many other configurations that interval graphs do not contain as well.

A graph class $\mathcal{C}$ is called *hereditary* if it has the property that whenever $G \in \mathcal{C}$, every induced subgraph H of G is also in $\mathcal{C}$. The property of being H-free is a hereditary property. Every hereditary graph class has a forbidden induced subgraph characterization, although the list of forbidden induced subgraphs may not always be finite.

1.3.1 Cluster Graphs

A cluster graph is a disjoint collection of cliques. That is, every connected component is a maximal clique.

If an edge xy of a graph denotes a symmetric relation between x and y, then the graph on a set of vertices is a cluster graph if and only if the relation is transitive. Cluster graphs have been used in a variety of applications whenever clustering of objects is studied or when consistent data is sought among noisy or error-prone data.

A graph G is a cluster graph if and only if G is P_3-free. Note that the cluster-completion problem in which one is allowed to only add edges to a given graph is polytime solvable since the only solution would be to add all possible edges within each connected component.

The cluster-deletion problem is NP-complete [130] [111] and hard to approximate [130].

The cluster-editing problem has been proven hard several times independently and in different contexts [91] [130] [5]. Both these problems were again proven to be NP-hard in [86] where they were further showed to remain hard for bounded-degree graphs and studied under alternate parameterizations.

Indeed, as cluster graphs give a simple and fundamental way of partitioning a network into clusters, the corresponding problems have been studied for their algorithms and alternate parameterizations as well. The fastest-known runtime for cluster-editing has been repeatedly improved [65], [121], [121], [71], [22], with the current best runtime standing at $O(1.62^k+m)$ [10]. Cluster-deletion can be solved in $O(1.415^k + n^3)$ [12].

For some applications where enforcing cliques to be completely disjoint from one another is too stringent, generalizations to clusters graphs have been studied. For instance, allowing two cliques to share at most t vertices or t edges was studied in [35]. Another is that of relaxing each component's structure from a clique to a k-plex [72]. The P_3-free characterization of cluster graphs allows for other natural generalizations, such as enforcing P_4-freeness or more generally S-free graphs where S is any collection of graphs which contain a P_3.

1.3.2 Quasi-Threshold Graphs

A *rooted forest* is a disjoint union of trees, each tree having a designated root vertex. The root should be thought of as a "top level" node, its children as second-level nodes, and so on. Define *reachability* in a tree between two nodes to mean that u reaches v (or v is reachable from u) if and only if the unique path from the root vertex to v passes through u. That is, v is reachable from u if u is an ancestor of v (equivalently, v is a descendant of u). The *comparability graph* of a rooted forest is a new graph with the same vertices as the rooted forest such that an edge is joined two vertices if one is reachable from another.

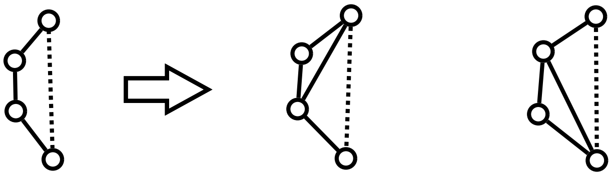


Figure 1.1: Wolk's characterization for quasi-threshold graphs states that if there are four vertices having at least those edges shown in the graph on the left, then it must be that one of the extra edges – shown in the situations on the right – must exist. The dashed line in the figure represents the fact that there may or may not be an edge there.

Wolk [144] introduced the concept of *comparability graphs of trees* and gave conditions on a simple undirected graph G which imply that G can be oriented in the directed tree structure described. Figure 1.1 depicts Wolk's formulation of the structural characterization for these graphs, and it is not hard to confirm that this is essentially equivalent to stating that there cannot exist an induced subgraph isomorphic to a P_4 or a C_4 in the graph.

The (P_4, C_4)-free graphs are now known as the class of *quasi-threshold graphs*[2] [25], *trivially perfect graphs*[3] [63], *comparability graphs of trees* [144], or *arborescent comparability graphs* [40].

A vertex v is called *universal* on a set of vertices S if v is adjacent to every vertex of S except possibly itself, in the case that $v \in S$.

Every quasi-threshold graph can be generated through a composition scheme that repeatedly performs any of two possible operations: (i) If G and H are two quasi-threshold graphs, then the disjoint union $G' = (V_G \cup V_H, E_G \cup E_H)$ is quasi-threshold; (ii) if G is quasi threshold, then $G^+ = (V_G \cup \{v\}, E_G \cup \{vx \mid x \in V_G\})$ is also quasi-threshold (that is, adding a universal vertex to a quasi-threshold graph yields a quasi-threshold graph).

Reversing this composition scheme says that every connected component of a quasi-threshold graph has a universal vertex, and the removal of any set of vertices - which is equivalent to taking an induced subgraph - also has the property that each of its connected components has a universal vertex.

A generative process for quasi-threshold graphs which adds one vertex at a time also exists. This generation scheme is used by Chu [23] to decide, in linear time, whether a given graph is quasi-threshold and in the case that it is not, the algorithm will produce a P_4 or a C_4 to certify this fact. The generative process uses one operation: add a new vertex v adjacent to any existing vertices $u_1, \ldots, u_t$ (possibly none) and further attach v to the connected component of each u_i. To use this as a recognition algorithm, we note the a vertex of largest degree can always be assumed to have been the last vertex added to a graph in this generation scheme, and a lexicographic BFS search can verify all the necessary adjacencies in linear time.

1.3.3 Cographs

Many graph properties can be deduced from looking at individual connected components of a graph separately. In this sense, a graph *decomposes*

[2] The term *quasi-threshold* comes from the fact that these graphs generalize *threshold* graphs, which are graphs created from weighted vertices, and two vertices u and v are joined by an edge if and only if the sum of the weights of u and v is above a given fixed threshold value. Threshold graphs are equivalently characterized as $(P_4, C_4, \overline{C}_4)$-free.

[3] A graph was defined to be *trivially perfect* if every maximal clique intersects the maximum independent set of the graph. The name derives from the fact that these are easily shown to be a subclass of *perfect graphs* which are an important class in algorithmic graph theory. It turns out that a graph is trivially perfect if and only if it is (P_4, C_4)-free.

into its connected components. Some properties are further maintained under a decomposition into connected components of the complement of the graph. When a graph G can decompose completely via connected components either in G or connected components in the complement $\overline{G}$, the graph is said to be a *complement-reducible graph* or a *cograph.* Cographs have been another important class of graphs to the development of algorithmic graph theory. The decomposition scheme defining the class of graphs has been generalized to apply to general graphs in the form of the *modular decomposition* and further as the *primeval decomposition* and *homogeneous decomposition.*

While it is not obvious from the above definition, cographs can also be characterized as P_4-free graphs. These serve as a generalization to P_3-free graphs in that they allow P_3s to exist in the graph provided the P_3 does not extend further to a P_4.

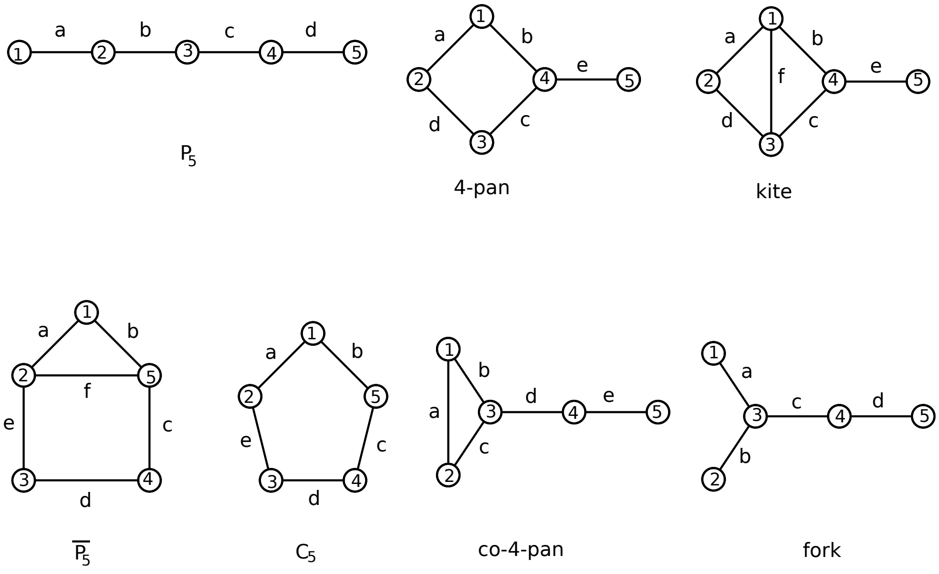
Cographs can be recognized in linear time [31]. The vertex deletion problem for cographs is NP-complete [94], and the edge deletion problem is also NP-complete [43]. Since cographs are self-complementary, the edge deletion problem is equivalent to the edge addition problem for this class. The problem of edge editing (allowing for both deletions and additions) has also been determined to be NP-complete [96].

The finite forbidden induced subgraph characterization for cographs implies that the above-mentioned NP-hard problems are fixed-parameter tractable. The problems of modifying a graph to a cograph has also been studied in the context of kernelizations [69].

Cographs have been generalized in a number of ways: P_4-sparse graphs were defined in a way that generalizes the forbidden induces subgraphs [78] while still being *perfect graphs* and recognizable in linear time [79], but they also generalize cographs through a decomposition scheme [80]. These have further been generalized to (q, t) graphs, which are those graphs for which every set of q vertices induces at most t P_4s. Cographs are $(4, 0)$-graphs and P_4-sparse are $(5, 1)$-graphs. Cographs are also exactly the graphs of *clique width* 2 [32], and bounded clique width graphs are known to have important algorithmic properties.

1.3.4 P_4-sparse Graphs

One generalization to the class of cographs is created by allowing P_4s to exist in a graph but in restricted amounts. Hoáng [78] introduced P_4-sparse graphs to be those for which every induced subgraph on five vertices induces at most one P_4. This immediately implies a forbidden induced subgraph



P_5 4-pan kite

$\overline{P_5}$ C_5 co-4-pan fork

Figure 1.2: The forbidden induced subgraphs for P_4-sparse graphs.

characterization which restricts any subgraph of five vertices inducing two or more P_4s. We include these graphs in Figure 1.2.

A special graph structure called a *spider* [79] commonly occurs in graph classes of bounded cliquewidth. We define two types of spiders here:

Definition 1.1. A graph $G = (V, E)$ is a *thin spider* if V can be partitioned into K, S and R such that:

i) K is a clique, S is a stable set, and $|K| = |S| \geq 2$.

ii) every vertex in R is adjacent to every vertex of K and to no vertex in S.

iii) each vertex in S has a unique neighbour in K, that is: there exists a bijection $f : S \to K$ such that every vertex $k \in K$ is adjacent to $f(k) \in S$ and to no other vertex in S.

A graph G is called a *thick spider* if $\overline{G}$ is a thin spider. Note that the vertex sets K and S swap roles under graph complementation, that condition (i) and (ii) hold for thick spiders, and that statement (iii) changes to saying that every vertex in S has a unique *non-neighbour* in K. The sets K, S and

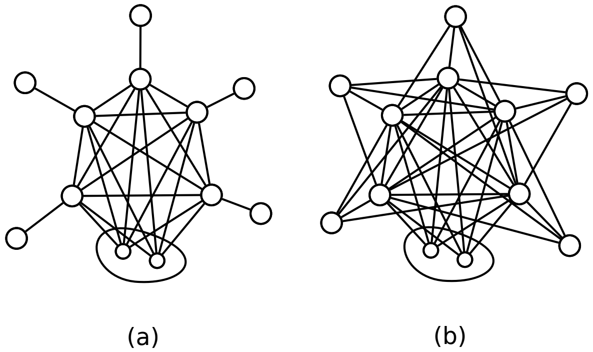


Figure 1.3: A thin (a) spider and a thick (b) spider with $|K| = |S| = 5$ and $|R| = 2$.

R are called the *body*, *feet* and *head* of the spider, respectively. The edges with one endpoint in S are called *thin legs* or *thick legs* for thin spiders or thick spiders, respectively. Examples of spiders are given in Figure 1.3.

Hoàng [78] defined a graph G to be P_4-sparse if every induced subgraph with exactly five vertices contains at most one P_4. The following decomposition theorem for P_4-sparse graphs was proven in [79]:

Lemma 1.2. *[79] Let G be a P_4-sparse graph. Then at least one of the following is true:*

i) G is disconnected

ii) $\overline{G}$ is disconnected

iii) G is a thin spider

iv) G is a thick spider

1.3.5 Chordal Graphs

Chordal graphs have been studied under the names *rigid circuit graphs* [39] or *triangulated graphs*. They are defined as graphs in which every cycle of size 4 or more contains a chord. Equivalently, chordal graphs are those containing no induced cycles of size 4 or more. They have been instrumental in the development of algorithmic graph theory, as *lexicographic breadth first search* (or LBFS) was first used to recognize chordal graphs in linear time, and LBFS has come to be a very important tool in algorithmic graph theory. Chordal graphs also naturally give rise to the important decomposition scheme known as the *clique separator decomposition* [134] [143], which decomposes graphs while maintaining certain structures.

Every cluster graph (defined in 1.3.1) and split graph (defined in 1.3.6) is chordal, and in fact the class of split graphs are exactly the graphs which are chordal and whose graph complement is also chordal. The importance of graph modification problems on chordal graphs has grown proportionately to the importance of treewidth which is a key graph invariant used heavily in algorithm design. Edge deleting [112], edge completing [149], edge editing [111] and vertex deleting [94] a general graph to a chordal graph is NP-complete. These were also all shown to be fixed-parameter tractable problems [104].

For a set of sets $\mathcal{S}$, an *intersection graph of* $\mathcal{S}$ is a graph where each $v \in V$ is an element of $\mathcal{S}$ and an edge uv is in the graph if and only if the sets u and v have nonempty intersection. An example of this is when $\mathcal{S}$ is a set of intervals on the real line. The resulting intersection graph of $\mathcal{S}$ is called an *interval graph*. Not every graph is an interval graph. One small example of a graph which is not an interval graph is a C_4. For all possible $\mathcal{S}$, the resulting set of interval graphs is called the *class of interval graphs.*

A chordal graph can be thought of as a generalization of *interval graphs.* An early characterization of chordal graphs shows that chordal graphs are exactly the graphs which are intersections of subtrees of a tree [59]. While this easily shows that interval graphs are a subclass of chordal graphs, the forbidden induced subgraph characterization of interval graphs is not easy to describe [93].

1.3.6 Bipartite and Split Graphs

A graph is *bipartite* if it can be partitioned into two sets in such a way that every edge of the graph has an endpoint in each partition. Bipartite graphs arise from applications where vertices usually represent two different types of objects. Some social networks are *affiliation networks* where a set of actors are related through a set of affiliations. When an affiliation network is created with affiliations and actors as vertices, and an edge representing when an actor is associated with a certain affiliation, the resulting network has a bipartite structure with one partition of vertices being the actors and the other partition being the affiliations [139]. Deciding whether a graph is bipartite, and finding the vertex bi-partition, can be solved in linear time.

A bipartite graph can also be described as graphs which can be partitioned into two stable sets. A *split* graph is a graph which can be partitioned into a clique and a stable set.

Bipartite graphs do not have a finite forbidden induced subgraph characterization: they are the graphs with no induced odd cycles. Modifying a graph to a bipartite graph through graph edits have been studied under the name of *odd cycle transversals* and the first fixed-parameter tractable algorithm for this problem [124] is credited with the initiation of the *iterative compression* method for algorithm design.

Split graphs do have a finite forbidden induced subgraph characterization: a graph is split if and only if it is $(2K_2, C_4, C_5)$-free, where a $2K_2$ is the graph on four vertices and two disjoint edges. Interestingly, the problem of modifying a graph to a split graph is NP-complete when dealing with only edge deletions or only edge additions [111], but it is polynomial-time solvable when both of these operations are allowed [73]. Deciding whether a graph is split, and finding a corresponding clique and stable set partition, can be solved in linear time. This algorithm uses just the degree sequence of the input graph to decide where to add and where to remove edges. The edge-edit distance to a split graph is called the *splittance* of a graph (so split graphs have splittance 0).

The edge editing problem to a bipartite graph is easily seen to be equivalent to the edge-deletion problem to bipartite graphs, since adding edges can never help make a graph bipartite. The edge deletion problem to bipartite graphs, known as *edge bipartization*, is NP-complete [147].

Chapter 2

Social Communities

2.1 Existing Methods for Cluster Partitioning

Some community-finding methods exploit the fact that as much a community should be densely connected within itself, it should not be as heavily connected to the rest of the network. There are several examples of definitions that require exterior sparsity in addition to interior density. In 1969, this idea was expressed in an *LS-set* which is a set S such that every vertex in S has more neighbours in S than in $G - S$ [99].

Many methods have been developed to identify dense clusters in networks. The measure of *betweenness centrality* [53] has been used by [62] to identify cohesive groups. The strategy in the Girvan-Newman algorithm is to identify and remove edges of high betweenness centrality (see Definition 4.3) since such edges are typically regarded as being edges that cross between separate communities. In the case of multiple choices of edges with equal and highest centrality, any arbitrary choice of these suffices. Upon deleting an edge, the algorithm recomputes the centrality of all edges in order to find the next edge to be deleted. If an edge deletion results in partitioning the graph into more connected components than there were before the edge deletion, the modularity of the network is measured. Once all edges have been removed in this manner, the step of this process which resulted with the largest modularity score gives a natural partition of the network into groups. The process runs in polynomial time and has been shown to produce meaningful results on real networks; however, its focus on edges which are not in communities does not imply or suggest a structure of community that we are after.

The Girvan-Newman algorithm is an example of a method that successively partitions a network via edge deletions. Other methods, such as partitioning along a minimum cut of a network [150], also yield a hierarchical decomposition of components of a network while removing multiple edges at once.

Algorithm 1: High-level description of the Girvan-Newman process for cluster-finding.

```
Algorithm GirvanNewman($G$):
Input: A Graph $G = (V, E)$
Output: A hierarchical decomposition of $G$

while ($G$ has at least one edge) do
    Let $e = xy$ be an edge of largest betweenness centrality;
    $G \leftarrow G - e$;
    if $x$ and $y$ are in different components then
        Record the new connected components;
    end
end
```

2.1.1 An Induced Subgraph Variation

Other algorithms, similar to the Girvan-Newman algorithm, have been suggested with *betweenness centrality* swapped out for another measure. For example, [122] observes that edges that cross dense groups are not typically in as many triangles as edges inside the dense clusters. Removing edges that appear in the fewest triangles results in partitions similar to those found by the Girvan-Newman method. These methods are also shown to generalize

LS-sets.

Algorithm 2: High-level description of Radicchi et al.'s process for cluster-finding.

```
Algorithm Radicchi($G$):
Input: A Graph $G = (V, E)$
Output: A hierarchical decomposition of $G$

while (G has at least one edge) do
    Let $e = xy$ be an edge involved in the fewest number of triangles
    (breaking ties arbitrarily, if needed);
    $G \leftarrow G - e$;
    if $x$ and $y$ are in different components then
        Record the new connected components;
    end
end
```

Many of the methods which use a structural definition for community and seek out these structures in networks result in problem formulations which are inherently NP-complete. Examples of this approach include clique, clan, club, and k-plex finding, and these are defined in the next section. While being NP-Complete is a computational obstruction, there are a variety of algorithmic techniques that can be used to extract these desirable structures from networks. For instance, Integer LPs such as those in [3] offer an exact algorithm for these problems, and also lend themselves readily to faster approximation algorithms. Fixed-parameter tractability (FPT) is another algorithm-design technique that can sometimes be used to solve an NP-complete problem with reasonable time. Finding network clusters via cluster editing, for example, has been studied extensively in the FPT framework [11] with great success.

The literature on finding cohesive subgroups of networks is vast, and methods such as spectral methods [131] and probabilistic model-fitting [75] have been explored. Many such methods are summarized in the surveys by [50] and [127].

2.2 Cliques and Beyond

In data where relationships (edges) between objects are expected to be transitive, the resulting graph that represents that relationship is expected to arrange itself into a disjoint union of cliques. These graphs are known

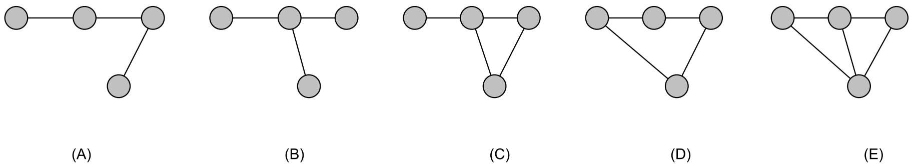


Figure 2.1: The one-vertex extensions of a P_3: (A) P_4; (B) claw; (C) paw; (D) C_4; (E) diamond.

as *cluster graphs* and they form a hereditary class of graphs which can alternately be characterized as the P_3-free graphs. We can thus say that this P_3-free *local structure* on three vertices completely characterizes the network structure.

When data-gathering methods are incomplete, the resulting network will be close to a cluster graph but with some missing edges. In other applications where false-positives appear, the corresponding graph will contain extraneous edges between the implied components. The common approach taken to identify the implied clusters of such networks is through graph editing: the addition or removal of as few edges as possible in order to obtain the desired structure. The associated algorithmic problem is known as *Cluster Editing*.

The idea of cluster editing is also known as *correlation clustering* [5] in machine learning and other fields of computer science and approximation algorithms have been designed for the problems of partitioning a network to minimize the number of inter-cluster edges and/or maximize the intra-cluster edges.

As clique components are a very stringent condition to impose on a social collection, various generalizations to cliques have been explored in the literature. Early attempts include that of Luce's *n-cliques* [100] and Alba's *n-clans* and *n-clubs* [2] [106]. An *n-club* of a network is an induced subgraph having diameter n. A clique is exactly a 1-club. The computational problems of determining whether a graph contains an n-clique or n-club were shown to be NP-complete in [4]. In fact, the problem of finding a 2-club was shown to be NP-complete even on relatively simple input graphs, such as those which are bipartite after deleting one vertex and graphs which are covered by 3 cliques [74].

Another generalization of cliques is the k-plex, which is a collection of n vertices in which every vertex is adjacent to at least $n - k$ other vertices in the collection [129]. Finding a k-plex of size n was shown to be NP-complete by [3]. Cluster graphs have also been generalized in such a way that cliques - rather than being completely disconnected - may intersect in at most s vertices or t edges [48]. Such graphs admit a finite forbidden induced subgraph characterization as well: for example, the graphs in which cliques are allowed to intersect in at most one vertex are exactly the diamond-free graphs, where the diamond is the one-vertex extension of a P_3 depicted in Figure 2.1. Again, the structure of this class of graphs is characterized via a characterization of its local structure.

The class of cluster graphs can be generalized through its local structure by allowing P_3s to exist in a graph, but forbidding some of the possible extensions of a P_3. A complete set of isomorphically-distinct one-vertex extensions of a P_3 are given in Figure 2.1.

2.3 Cluster Deletion

Cluster graphs have been used in a variety of applications whenever clustering of objects is studied or when consistent data is sought among noisy or error-prone data [5].

The parameterized Cluster Deletion problem takes a graph G and an integer k as input and asks whether it is possible to delete at most k edges F from G in order to make $G \setminus F$ a cluster graph. This problem is known to be NP-hard [130].

Many NP-complete problems have been studied on restricted input in order to find classes of instances for which efficient solutions to the problem can be found. For example, the NP-complete problem Max Clique can be solved in polynomial time on perfect graphs [68], in $O(n^3)$-time on weakly chordal graphs [77], and in $O(m+n)$-time on chordal graphs [59] and on cographs [31] and a host of other graph classes (for example, [26], [60], [16]). Clique remains hard on (C_5, P_5)-free graphs, and even on the relatively small class formed by the complements of square-and-triangle-free graphs [64].

A general result of Cai [19] on graph modification problems implies that Cluster Deletion(G, k) is a *fixed-parameter tractable* problem, solvable in $O^*(2^k)$ time, where the notation $O^*(f(n,k))$ means $O(p(n,k)f(n,k))$ for some polynomial function p (n is the number of vertices of G). This standard search method, finds a P_3, and then branches on the two possible edge-deletions. An improved branching method was developed by Gramm et al. in [65], improving the runtime to $O^*(1.77^k)$. Later, the same authors developed an algorithm for Cluster Deletion running in $O^*(1.53^k)$ time.

There have been further improvements on the above runtime using algorithms that rely on the property of being able to solve Cluster Deletion efficiently (i.e. in polynomial time) when exploiting certain structure. Damaschke [36] characterized the structure of graphs in which no edge is contained in three P_3s and used this structure to optimally delete all remaining P_3s in polynomial time. Using branching rules to reduce a general instance to this special structure resulted in an algorithm running in $O^*(1.47^k)$ [36]. This has been improved even further to $O^*(1.415^k)$ by Böcker and Damaschke [12] using the fact that an optimal cluster deletion set can be found in polynomial time on graphs which are formed as a clique and a constant number of other vertices attached to the clique.

In light of these results, it is of interest to determine any significant subclasses of input graphs on which Cluster Deletion can be solved in polynomial time. For instance, the edge-deletion problem which asks how to delete the fewest number of edges from a graph in order to obtain a cograph was studied in [108] where improved algorithms with given by first deleting to a certain superclass of cographs, and then deleting remaining edges in polynomial time.

In this section, we present an algorithm that solves Cluster Deletion in polynomial time on cographs. This serves to fill Phase 2 of a general meta-algorithm (Algorithm 5) that is the focus of Chapter 5. We also observe that this result is close to the boundary of polytime solvability by noting that Cluster Deletion remains NP-hard on a slightly larger graph class. As far as we know, this is the first published algorithm that solves Cluster Deletion in polynomial time on a well-studied graph class.

The literature on finding subclasses of graphs on which certain NP-Complete problems become polytime solvable is vast, particularly with respect to clique finding and vertex colouring. Graph modification problems, however, have not been extensively studied on graph classes, perhaps with the exception of the *chordal completion* or *treewidth* problem. Chordal completion is also known as the *minimum fill-in* problem, and examples of graph classes on which minimum fill-in is solvable in polynomial time are chordal bipartite graphs [20], circle graphs and circular-arc graphs [84], and house-hole-domino-free (HHD-free) graphs [17].

2.3.1 On the Hardness of Cluster Deletion

The proof of Natanzon [111] reduces from the NP-Complete Clique problem to Cluster Deletion by first considering a general instance G of the Clique problem and completely joining a clique to it. We observe that as long as the initial instance of G is NP-hard, the class of constructed graphs obtained will be hard instances for Cluster Deletion. Some example classes of where Clique remains hard is on (C_5, P_5)-free graphs and on $(2K_2, 3K_1)$-free graphs.

Call a vertex *universal* on a subgraph H if it is adjacent to every vertex in H (except itself).

Lemma 2.1. *Let graph G have no induced subgraph isomorphic to F, where F is a subgraph without a universal vertex. Then the graph G^+ obtained in Natanzon's construction (G completely joined to a new clique) is also F-free.*

Proof. Assume on the contrary that G^+ contains some induced subgraph H isomorphic to F. Since G is F-free, H must contain at least one vertex from the clique joined to G. But every vertex in this clique is universal on G^+ and so must be universal on F, a contradiction. ■

Corollary 2.2. *Let $(F_1, F_2, F_3, \ldots)$ be a sequence of graphs such that F_i does not contain a universal vertex for each i. Then* Cluster Deletion *remains NP-hard on the class of $(F_1, F_2, F_3, \ldots)$-free graphs.*

Since each of $C_5, P_5, 2K_2, 3K_1$ does not have a universal vertex, it follows that the constructed instances of Cluster Deletion from Natanzon's proof are also free of these subgraphs when the original Clique instance is. Thus we have:

Corollary 2.3. Cluster Deletion *remains NP-hard on (C_5, P_5)-free graphs and on $(2K_2, 3K_1)$-free graphs.*

Shamir et al. [130] showed that Cluster Deletion is still hard when enforcing that the input graph be deleted into exactly $d \geq 3$ components. They also showed that when deleting to exactly $d = 2$ components, the problem is polynomial time solvable.

Although [87] [86] prove that Cluster Deletion is hard for graphs with maximum degree 4, it also gives a $O(n^{1.5} \log^2 n)$ polynomial time algorithm solving Cluster Deletion on graphs with maximum degree 3.

The above results explore the boundary of where the Cluster Deletion goes from being NP-hard to being polynomial time solvable. In light of the results which use polynomial time solvability of certain subclasses in the design of general exponential-time algorithms, classifying the polynomial time instances of Cluster Deletion proves to be of great algorithmic importance while additionally being an interesting investigation in its own right.

The Clique problem is known to be polynomial time solvable on perfect graphs [68] and thus all of its subclasses. As mentioned earlier, Clique remains hard on (C_5, P_5)-free graphs and thus Cluster Deletion is hard on

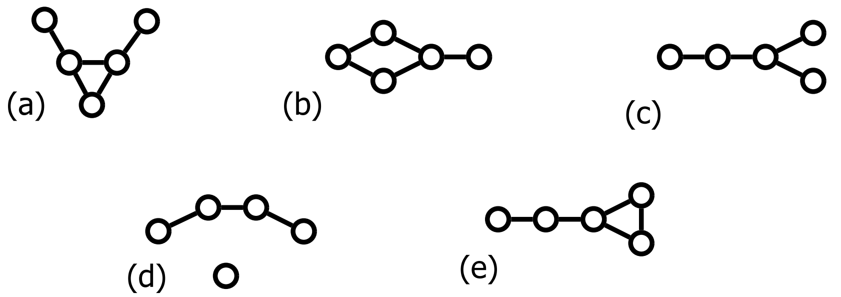


Figure 2.2: (a) bull; (b) 4-pan; (c) fork; (d) gem; (e) co-gem; (f) co-4-pan.

this graph class as well. Another class (call it $\mathcal{Z}$-class) defined by forbidding all of (C_5,P_5, bull, 4-pan, fork, co-gem, co-4-pan) is a superclass of cographs (see Figure 2.2). This class restricts 7 of the 10 possible ways a vertex can join a P_4. Since every forbidden graph in the list defining the $\mathcal{Z}$-class has no universal vertex, the following lemma shows that Cluster Deletion remains hard on $\mathcal{Z}$-graphs.

Lemma 2.4. Clique *remains NP-hard on class $\mathcal{Z}$.*

Proof. Let G be a graph with m edges. Solving independent set on G is NP-complete. Following Poljak [120], construct graph F which replaces every edge of G with a 3-edge path. This is called a 2-subdivision of G. Note that $\alpha(F) = \alpha(G) + m$. So finding $\alpha(F)$ will find $\alpha(G)$ and shows that independent set is NP-complete on 2-subdivision graphs. Now a 2-subdivision cannot contain a C_5, 4-pan, co-4-pan, bull, gem, co-fork, or a co-P_5 as an induced subgraph. So 2-subdivisions are a subclass of co-$\mathcal{Z}$, and so independent set is NP-complete on co-$\mathcal{Z}$. Therefore clique is NP-complete on $\mathcal{Z}$.[4] ■

2.3.2 Cluster Deletion on Cographs

A *clique partition* of a graph G is a partition $CP = \{V_1, V_2, \ldots, V_t\}$ of the vertex set such that each $V_i \in CP$ induces a clique in G. Note that Cluster Deletion can be rephrased to ask for a clique partition such that the sum of the number of edges that join V_i to V_j over all $i \neq j$, is a minimum. For this reason, Cluster Deletion has also been studied under the name of *the minimum edge clique partition problem* [38]. A *greedy max*

[4] Using the same reasoning, the result remains true if the graphs co-C_6, co-C_7 and co-C_8 are added to $\mathcal{Z}$.

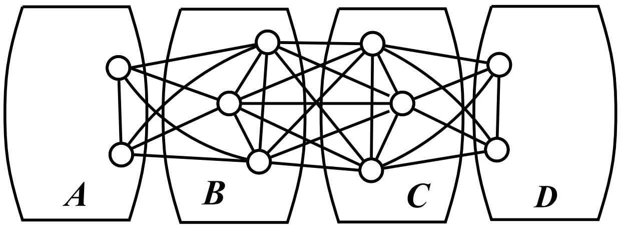


Figure 2.3: A C_4-free graph for which an optimal min edge cluster partition does not contain the maximum clique in a single part of the partition.

clique partition of a graph G is a clique partition $\{A_1, A_2, \ldots, A_k\}$ where A_1 is a maximum clique of G, and A_i is a maximum clique of $G \setminus \bigcup_{j=1}^{i-1} A_j$.

Note that greedily selecting maximum cliques does not necessarily yield a minimum edge clique partition in general. Figure 2.3 depicts a graph which is a $(\overline{K}_3, C4, C5)$-free graph where choosing the unique maximum clique (which is induced by the set $B \cup C$ in the diagram) and then the remaining two cliques (one induced by A and the other induced by D) yields a clique partition which realizes the cluster deletion solution of deleting the 6 edges joining A to B and the 6 edges joining C to D, for a total of 12 edge deletions. However, $A \cup B$ is a clique and $C \cup D$ is a clique, and choosing those two cliques is equivalent to a solution to the cluster deletion problem which deletes the 9 edges joining B to C.

The $(\overline{K}_3, C4, C5)$-free graphs is a fairly small subclass of P_5-free graphs, so it is rather remarkable that we have the following theorem:

Theorem 2.5. *[56] If G is a P_4-free graph, then the minimum edge clique partitions of G are precisely the greedy clique partitions of G.*

The proof of this theorem is not included here but can be found in [56]. Although following the details of this proof is rather arduous, it results in a truly simple algorithm for solving cluster deletion on cographs.

2.3.3 Algorithms

We present here the simple algorithm for solving cluster deletion on cographs in polynomial time, as implied by Theorem 2.5.

Algorithm 3: Cluster deletion algorithm on cographs.

```
Algorithm ClusterDeletion($G$):
Input: A cograph $G = (V, E)$
Output: A set of edges $S \subseteq E$ such that $G - S$ is a cluster graph.

$S \leftarrow \emptyset$;
while $|V(G)| > 0$ do
    Choose a maximum clique $C$ of $G$;
    For every edge $cx$ with $c \in C$ and $x \in G - C$, add $cx$ to $S$;
    Remove $C$ from $G$;
end
return $S$;
```

This routine can be used to improve a bounded search tree method for solving Cluster Deletion on general graphs. In particular, it can be used as a realization of Phase 2 of Algorithm 5 (Chapter 5).

We illustrate a trace of this algorithm performed on a cotree representation of a cograph. For completion, we quickly describe cotrees here.

Definition 2.6. A *cotree* of a cograph G is a rooted tree representation T of G where every vertex of G appears as a leaf node of T and the internal nodes of T are labeled either 0 (for disjoint union) or 1 (for complete join). For every two vertices u, v of G, if uv is an edge then the lowest common ancestor node in T of the leaf nodes u and v is a 1 node, and is 0 in the case that u is not adjacent to v.

The 0-nodes can be thought of as a disjoint union of all the subgraphs rooted at the children of the 0-node. The 1-nodes can be thought of as a complete-join between all the subgraphs rooted at the children of the 1-node. See [31] for a linear time algorithm for constructing a cotree of a cograph.

To obtain a maximum clique of a cograph, we process the tree bottom-up, by first labeling each node with weight 1, and then every 0-node is labeled with the maximum weight of its children and every 1-node is labeled with the sum of the weights of its children. To make the cotree data structure more conducive to our need of finding maximal cliques, let a 0-node point towards its children achieving the maximum weight among all its children. The size of a maximum clique in a cograph is then the label assigned to

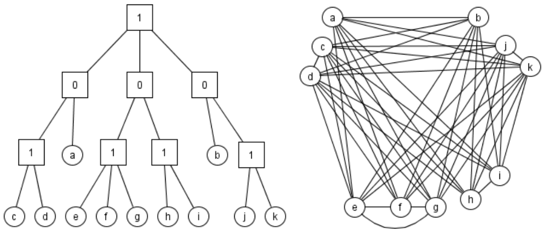


Figure 2.4: A cotree (left) and its corresponding cograph (right).

the root. To find the nodes of the maximum clique, from the root one can traverse down each branch from a 1-node and just a single branch that is oriented away from each 0-node, and all leaves reached are the nodes of a maximum clique.

A cotree can be updated when vertices are deleted from a cograph simply be deleting the corresponding leaf nodes of the cotree. When an internal node N of the cotree has 0 children, N can also be deleted. If N has a single child which is a vertex (leaf) x, then x can be made a child of the parent node of N and N can be deleted. If N has a single child which is another internal node N', then the children of N' can be made children of the parent node of N and both N and N' can be removed.

Since the internal nodes will always have 2 or more children, the number of internal nodes of a cotree will always be less than the number of leaf vertices, and so computing the maximum clique size labels is a linear time process.

Figure 2.5 shows the process of removing the maximum clique and reducing the cotree from a graph on 11 vertices and 45 edges. The first maximum clique found is of size 7, thus containing $\binom{7}{2} = 21$ edges in it. Once the maximum clique is extracted and the cotree is reduced, we find a clique of size 4 containing 6 edges. This implies the minimum size of a set of edges solving Cluster Deletion on this graph is of size $45 - 21 - 6 = 18$.

When using a bounded search tree techniques (see Chapter 5), we bound and prune the search tree based on the numerical size of the solution, and so computing the number of edges not involved in the greedily-extracted maximum cliques (by arithmetic) is all that is needed to use this algorithm as a subroutine for a general search. Constructing the actual edges in the solution set would require multiple adjacency tests, each taking $O(\log(n))$ time.

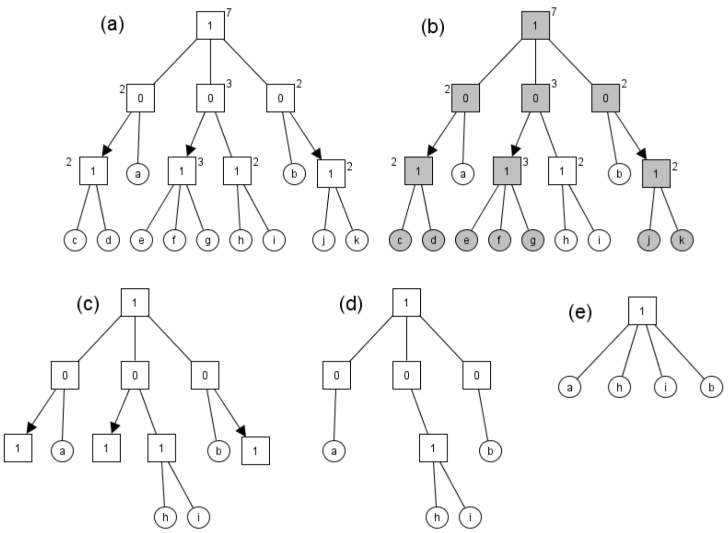


Figure 2.5: (a) The original cotree with labels on the internal nodes indicating the max clique size below it. (b) Identifying the nodes of a maximum clique from the root by following oriented edges downward. (c) The cotree after the vertices of the maximum clique are removed. (d) The cotree after removing interval nodes with 0 children. (e) The cotree after removing internal nodes with 1 child.

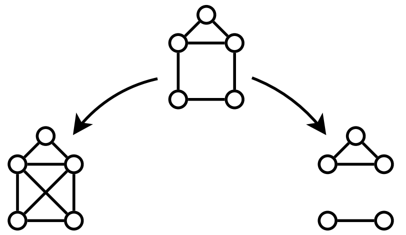


Figure 2.6: Two equally-weighted outcomes of modifying a graph to a closest (P_4, C_4)-free graph.

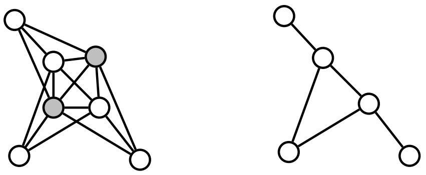


Figure 2.7: A community satisfying Freeman's transitively overlapping clique condition (left) that is not hereditary. Removing the two filled vertices yields a graph (right) which no longer satisfies the definition.

2.4 Quasi-Threshold Graphs as Communities

We define a new community structure by generalizing cluster graphs through local structure. Rather than each community being a clique as in the case of cluster graphs, we relax that definition so that our components are family-like structures. As in the case of cluster editing where communities are found by finding a closest P_3-free graph, we find *familial groups* of a network by finding a closest (P_4, C_4)-free graph.

Definition 2.7. The familial groups of a network G are the connected components of a closest (P_4, C_4)-free graph.

The measure of what a "closest" network can vary, but following the paradigm of the cluster editing problem, we shall use the measure of total edge additions plus edge deletions. These are collectively referred to as *edge edits.*

Since there are several ways to "destroy" a P_4 or C_4 with edge edits, the resulting decomposition may not be unique (but this is a reality in P_3-editing for correlation clustering, and many other community-finding methods as well). An example of two different outcomes of editing a given graph with an equal number of edge edits is shown in Figure 2.6. Under a framework of weighted modifications, for instance, a cost of α for adding an edge and β for deleting an edge, one could weigh one decomposition better than another. For instance, if we were interested more in seeing how a network decomposes into groups, we would set $\alpha > \beta$, and vice versa if we were more interested in

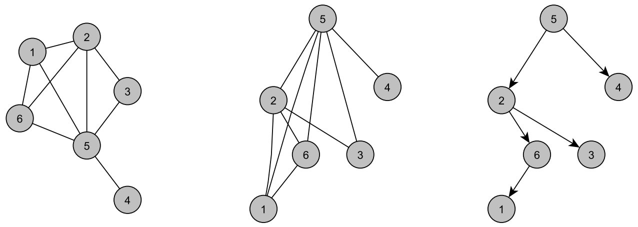


Figure 2.8: (left) A quasi-threshold graph; (center) A tree arrangement of the graph on the left; (right) The comparability tree of the quasi-threshold graph on the left

seeing how individuals in a community are organized. Weights on individual edges with perturbations can ensure a unique optimal decomposition into familial groups if desired. Here, we only consider $\alpha = \beta = 1$ and we are interested in modifying networks using as few modifications as possible.

2.4.1 Properties of Familial Groups

In the following subsections, we give support to the idea that a quasi-threshold graph (a (P_4, C_4)-free graph) is an *ideal structure* for networks composed of family-like or hierarchically-organized communities.

Hierarchical Representation of Familial Groups

Quasi-threshold graphs have many characterizations and properties (see Section 1.3.2 for details) but we will be mainly interested in the forbidden induced subgraph characterization and the rooted tree representation of quasi-threshold graphs, which we describe here.

Every quasi-threshold graph G can be arranged into a forest-like structure (a set of tree-like structures) in which every vertex is adjacent (in G) to every descendant in the tree. In particular, the root of a tree T is adjacent (in G) to every vertex in T, and there does not exist an edge joining two vertices in separate trees. An example of a quasi-threshold graph and its associated comparability tree are given in Figure 2.8. Note that every leaf in a tree is adjacent to all of its ancestors and that every set of vertices along a root-to-leaf path forms a maximal clique of the graph.

In the graph in Figure 2.8, vertex 5 is a universal vertex (a vertex adjacent to every vertex). It is the root of the associated tree. The rest of the vertices form two connected components: $\{4\}$ and $\{1, 2, 3, 6\}$. In the component $\{1, 2, 3, 6\}$, vertex 2 is universal and is the root of the subtree consisting of vertices 1,2,3, and 6. The other subtree consists of a single vertex 4. In the subtree rooted at 2, the positions of 1 and 6 can be interchanged with each other arbitrarily because they are structurally equivalent. In this example, the whole network is a familial group. If vertex 5 is removed from the network, then we will have two separated smaller familial groups $\{1, 2, 3, 6\}$ and $\{4\}$. In turn, after removing vertex 2 in the group $\{1, 2, 3, 6\}$, we will get two even smaller familial groups $\{1, 6\}$ and $\{3\}$.

Quasi-threshold graphs are natural structures that arise from modeling certain applications. For instance, if a graph is created on a set of species such that an edge is drawn between two species if and only if they have an ancestor/descendant relationship, then the graph created will form a quasi-threshold graph if the information obtained was error-free. Another example is that of a corporate structure in which every employee (except one) has a direct supervisor, and that commands can be passed to an employee from her supervisor or her supervisor's supervisor, etc. When an edge is joined between any two individuals on which a command can pass, the resulting graph is a quasi-threshold graph.

Familial Groups as Robust Communities

Freeman [54] gave a definition for social community which uses the idea of *clique overlaps*. Two cliques overlap if they intersect in at least one vertex. The definition [54] can be summarized as follows: a set of maximal cliques $C_1, C_2, C_3, \ldots, C_k$ which induces a connected graph forms a community if the cliques C_i overlap transitively. That is, for any three cliques C_i, C_j, C_k, if C_i overlaps C_j and C_j overlaps C_k, then C_i and C_k must also overlap. Freeman rationalized his definition by stating that an individual should be contained in a single community (that is, a network should decompose into disjoint communities), that it generalized cliques, and that it is applicable to networks in which only relationships (of unknown strength) between pairs of individuals was known. That is, his definition applies to undirected and unweighted graphs.

We will enforce a level of robustness to this definition of community to create a tighter definition of community. The removal of any vertices from a graph G leaves behind a graph H which is an induced subgraph of G. The robustness we impose can be stated as follows: a set of vertices S will form a familial group if S *and every connected induced subgraph of* S satisfies the above transitively-overlapping clique property. Socially speaking, the community remains intact if the removal of any number of individuals leaves the group connected. Or, in the case that some "important" individuals leave the community and disconnect it, then the remaining connected components will themselves form smaller communities. An example of a community which satisfies Freeman's transitively-overlapping clique property, but not hereditarily, is depicted in Figure 2.7.

We show that simply requiring Freeman's transitively-overlapping clique condition to be hereditary yields a formulation of social community which exactly corresponds to connected (P_4, C_4)-free graphs.

Theorem 2.8. *A connected set S of vertices satisfies Freeman's transitively-overlapping clique condition in every connected induced subgraph if and only if S induces a connected (P_4, C_4)-free graph.*

Proof.

If S satisfies the transitively-overlapping clique condition for every induced subgraph, then it cannot contain an induced path on 4 vertices $abcd$ since each edge is a maximal clique while ab overlaps with bc and bc overlaps with cd, but ab does not overlap with cd. Similarly, it cannot contain an induced cycle on 4 vertices $abcda$ for the same reason. So any graph satisfying the transitively-overlapping clique condition must be (P_4, C_4)-free.

Conversely, if a connected graph S is (P_4, C_4)-free, it must have a vertex u which is adjacent to all other vertices in S [146]. Since there is such a *universal* vertex u in every connected component of a (P_4, C_4)-free graph, every maximal clique in a connected component must include u, and so all maximal cliques in the connected component overlap, at least on vertex u. Consequently, the cliques overlap transitively. □

Familial Groups as an Extension of Triadic Closure

Some sociometric data not only measures when two objects are related, but also measures the strength of the tie between them. In 1973, Granovetter [67] formulated:

The Weak-Tie Hypothesis: if a is strongly tied to b and a is strongly tied to c, then it is more likely than not that b and c are at least weakly tied to each other.

Granovetter observed that the weak-tie hypothesis can be used to assert that the most unlikely triad to appear in a social group is when a is strongly tied to b and a is strongly tied to c, while b and c have no social relation between them. He goes on to propose a graph edit operation called *triadic closure* which adds at least a weak tie between b and c. In the framework of unweighted edges, this is exactly the condition that a social group is P_3-free as discussed previously.

We generalize the forbidden restriction on triads to forbidding certain configurations on 4 nodes, the P_4 (which contains two induced P_3s) and the C_4 (which contains four induced P_3s).

An intuitive argument further supports the restriction of long induced paths or cycles from social communities. A close-knit community should have relatively low diameter, and the existence of two vertices of geodesic distance d from each other would imply the existence of an induced path of length d. Thus a social community should be P_d-free from some relatively small value of d. An argument against the existence of induced 4-cycles in communities is that if a is tied to b, b to c, c to d, then d tied to a, it is highly likely that a will get to know c or b to know d. That is, ac and bd are highly-likely chords in the cycle $abcda$. As such, it is reasonable to expect social communities to be P_d-free and C_4-free for relatively small values of d.

While we will be concerned with (P_4, C_4)-free graphs here, larger and more relaxed communities could be identified if the focus is changed to (P_5, C_5)-free graphs or (P_5, C_4, C_5)-free graphs.

Friedkin's Horizon of Observability

In 1983, Friedkin [55] studied the *observability* between two actors of a network, loosely defined as whether one actor knows what the other actor is doing. He writes that in a social control system, observability is a prerequisite for control and that for nonadjacent nodes in such a network, control must be indirect and can exist if and only if an observer can gain information from a node some distance away and that reactions can be somehow transmitted through intermediate nodes.

Friedkin states two hypotheses: (i) that observability declines with distance between two nodes, and (ii) likelihood of observability between two particular nodes increases with the number of shortest paths between them.

2.5 Hardness of Finding Familial Groups

If G is a graph and S is a set for which $S \subseteq E(G)$, we use $G - S$ to mean the graph $(V(G), E(G) \setminus S)$. Similarly, when T is a set of vertex pairs which are not in G, we use $G + T$ to mean the graph $(V(G), E(G) \cup T(G))$. When S and T are disjoint sets of pairs of vertices, note that $G - S + T$ is equivalent to $G + T - S$. The computational problem of finding familial groups of a network is as follows:

Problem 4. **Quasi-Threshold Editing**(G, k): Given a graph G and an integer k, is there a set S of edge deletions and a set T of edge additions such that $|S| + |T| \leq k$ and $G - S + T$ is (P_4, C_4)-free?

The quasi-threshold edge-addition problem has been studied in [70] and the quasi-threshold edge-deletion problem in [108]. To our knowledge, the quasi-threshold editing problem has not yet been studied directly.

Given any graph as input, the algorithm of [23] decides in linear time ($O(m + n)$) whether the input is quasi-threshold and in the case that it is not, a P_4 or a C_4 will be produced. The computational status of the problem of finding the closest quasi-threshold graph (in terms of the number of edge modifications) was stated as an open problem in [18], and then in [102], and again in [97]. We resolve this open question by showing that this problem is NP-complete by observing some extensions to a theorem in [97].

Theorem 2.9. *Quasi-Threshold Editing is NP-complete.*

[43] proved that Cograph Deletion is NP-Complete by a reduction from Exact 3-Cover. [97] used the same construction to show that Cograph Editing is NP-Complete by strengthening the proof used for Cograph Deletion.

A quick description of the proof, without the details, is as follows: the reduction from Exact 3-Cover used by [97] to show that Cograph Editing is NP-complete constructs a graph G^* which is also C_4-free. The optimal edge-edit set for G^* that destroys all P_4s does not produce any C_4.

Since every quasi-threshold graph is a cograph, the number of edits required to the closest quasi-threshold graph is at least the number of edits required to obtain the closest cograph.

An algorithm solving quasi-threshold editing, applied to G^*, would destroy the P_4s (and not have any C_4s to worry about, as observed above) and would thus provide a solution to the instance of Exact 3-Cover.

The complete details of this proof are now provided.

Proof of Theorem 2.9.

We use the same construction and notation as those in [43], which was also used by [97].

Let $S = \{s_1, s_2, \ldots, s_n\}$ and $\mathcal{C} = \{S_1, S_2, S_3, \ldots, S_m\}$ be an instance of Exact 3-Cover. Since each set $S_i \in \mathcal{C}$ contains three elements from S, an exact 3-cover of S would use exactly $\frac{n}{3}$ sets from $\mathcal{C}$, as the Exact 3-Cover problem requires that each s_i is covered exactly once. We let $n = 3t$ and $r = \binom{3t}{2}$. Construct an instance of quasi-threshold editing as follows:

- Each s_i is a vertex, and the set S of these induces a clique.
- For every $S_j \in \mathcal{C}$, create two cliques X_j and Y_j such that $|X_j| = r$ and $|Y_j| = q$, where $q = 9(m - t)r + 3(r - 3t)$.

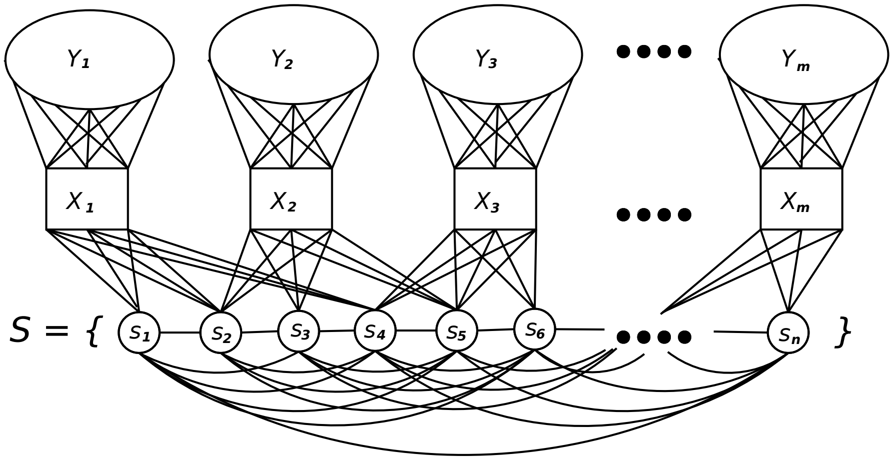


Figure 2.9: An instance of Quasi-Threshold Editing when reduced from an instance of Exact-3-Cover having $S_1 = \{s_1, s_2, s_4\}$, $S_2 = \{s_2, s_3, s_5\}$ and $S_3 = \{s_4, s_5, s_6\}$.

- Each of the three elements s_a, s_b, s_c of S_j is adjacent to every $x \in X_j$ and every $x \in X_j$ is adjacent to every $y \in Y_j$.
- No other edges exist in this graph.

The parameter to this instance of quasi-threshold editing is $k = \frac{q}{3} = 3(m-t)r + (r - 3t)$. This construction is depicted in Figure 2.9.

We note that if the instance of Exact 3-Cover is nontrivial (if some s_i exists in at least two 3-sets) this constructed graph is not a quasi-threshold graph since there are many P_4s, for instance, starting in some Y_j, adjacent to a vertex in X_j, adjacent to s_i, and adjacent to another X_k. There are no induced C_4s in this graph, however.

First, we prove that if we have a solution to the Exact 3-Cover instance, we can find at most k edge edits to turn this constructed graph into a quasi-threshold graph. Say that $\mathcal{C}'$ is a collection of t subsets of $\mathcal{C}$ such that the union of subsets in $\mathcal{C}'$ is S. For every pair s_i and s_j in S, delete the edge joining s_i and s_j if they do not coexist in a 3-set S_l in the solution $\mathcal{C}'$. These amount to $r - 3t$ edge deletions. Further, delete any edges from an X_i to S if S_i is not in $\mathcal{C}'$. This adds another $3(m-t)r$ deletions. In total, this gives $3(m-t)r + r - 3t = k$ edge edits, resulting in a (P_4, C_4)-free graph.

Note that a quasi-threshold editing set of size at most k is also a cograph editing set of size at most k. The same argument used in [97] can be used to show that the editing set contains edge deletions only. For the sake of completeness, we include the proof here.

Assume we have a quasi-threshold editing set E' of size at most k, where E' is a set of vertex pairs of G^* (if $uv \in E'$ is an edge of G^*, this represents an edge deletion; if $uv \in E'$ is not an edge of G^*, this represents an edge addition). The modified graph $G' = (V(G^*), E(G^*)\Delta E')$ is (P_4, C_4)-free, where Δ denotes the symmetric difference of sets. Call a vertex *affected* if it is a vertex with at least one incident edge that was modified by the k-edge edit set E'. Since each edge edit is incident on two vertices, there are at most $2k$ affected vertices.

Since $|Y_i| = 9(m-t)r + 3(r-3t) = 3k > 2k$, each Y_i set contains an unaffected vertex. We show that the edge edit set E' does not contain any edge additions.

Claim 1. E' contains no edge from $X_i \cup Y_i$ to $X_j \cup Y_j$.

Proof.
Assume there is an edge $u = v_i v_j$ from $X_i \cup Y_i$ to $X_j \cup Y_j$, with $i \neq j$ as $X_i \cup Y_i$ is already a clique. Then let $y_i \in Y_i$ and $y_j \in Y_j$ be unaffected vertices. Since v_i and v_j are affected by u, they are distinct from y_i and y_j. It is readily seen that $y_i v_i v_j y_j$ is a P_4, contradicting the fact that G' is quasi-threshold, so there can be no such edges. □

Claim 2. Every vertex s_i in G' is adjacent to vertices of at most one X_j.

Proof.
Assume s_i is adjacent to $x_p \in X_p$ and $x_q \in X_q$ where $p \neq q$. From the previous claim, x_p is not adjacent in x_q in G'. Let $y \in Y_p$ be an unaffected vertex. Then $y_p x_p s_i x_q$ is a P_4, contradicting the fact that G' is quasi-threshold. □

Claim 3. If in G' we have that s_i is adjacent to X_p and s_j is adjacent to X_q with $p \neq q$, then E' must delete the edge $s_i s_j$.

Proof.
Since the previous claim shows that each s_i is adjacent to at most one of the X_l, we have that s_i is adjacent to X_p and so cannot be adjacent to X_q. Similarly, s_j cannot be adjacent to X_p. Since the first claim shows there is no edge from X_p to X_q, we have a P_4 from X_p to s_i to s_j to X_q, unless $s_i s_j$ is a deleted edge. □

Claim 4. Every set X_i has neighbour s_j in the modified graph G'.

Proof.
The total number of edges in G^* joining $\bigcup X_i$ to S is $3rm$, and Claim 2

implies that the edited graph G' has at most rn edges which join $\bigcup X_i$ to S, which means that at least $3rm - rn = 3rm - 3rt = 3(m-t)r$ edges were removed by E'.

Since the size of the edit set is $|E'| \leq k = 3(m-t)r + r - 3t$ edges and we have described at least $3(m-t)r$ deletions in E', we have at most $r - 3t$ edits unaccounted for, which is less than the size of any X_i since $|X_i| = r$. Thus every X_i is adjacent to some $s_j \in S$. □

Corollary 2.10. *The number of edge edits which are not of the form sx for some $s \in S$ and $x \in \bigcup X_i$ is at most $r - 3t$. In particular, the number of edge edits in $\bigcup(X_i \cup Y_i)$ is less than $r = |X_i|$.*

Claim 5. If E' is a minimum edge edit set, then E' does not add any edge from Y_i to s_j.

Proof.
Assume there is some $y_i \in Y_i$ that is adjacent to s_j in the modified graph G'.

Firstly, assume that there is set $B \subseteq S$ of vertices s_b such that s_b is adjacent to s_j in G'. Then since every Y_i has an unaffected vertex y_i^*, we have a $P_4 = y_i^* y_i s_j s_b$ for each $s_l \in B$ and so $y_i s_b$ would also have to exist in E'. If this occurs, removing $y_i s_j$ and every $y_i s_b$ for each $s_b \in B$ from E' and adding the deletion $s_j s_b$ would decrease $|E'|$ by 1, contradicting the fact that $|E'|$ is a minimum.

Now we can assume s_j is not adjacent to any other $s_b \in S$ in G'. If s_j is adjacent to X_i, then the connected component containing s_j in the graph G' must be the induced graph on vertex set $Y_i \cup X_i \cup \{s_j\}$. So if edge $y_i s_j \in E'$, then this edge can be removed from E', yielding a smaller edge-edit solution, since the graph induced by $Y_i \cup X_i \cup \{s_j\}$ in G is already (P_4, C_4)-free.

On the other hand, if s_j is adjacent to some X_p where $p \neq i$ then consider an unaffected vertex $y_p \in Y_p$. Using a vertex $x_p \in X_p$ which is adjacent to s_j as well as y_p, we find the P_4 $y_p x_p s_j y_i$.

Finally, if s_j is not adjacent to any vertex of any X_l in G', then there is a P_4 $s_j y_i x_i s_q$ in the case that X_i is adjacent to s_q, or else $\{s_j\} \cup Y_i \cup X_i$ is a connected component in G', and the added edge from s_j to y_i can be removed from E' yielding a better edit set. (Note that there is some x_i for which $y_i x_i$ is not an edge deletion in E' by Corollary 2.10.) □

Claim 6. If E' is a minimum edge edit set of G such that the modified graph G' is quasi-threshold with $|E'| = k$, then E' has no edge additions.

Proof.

This follows from the previous set of claims, and we summarize these here.

There is no edge added from any $X_i \cup Y_i$ to any $X_j \cup Y_j$ by Claim 1. There is no edge added from any Y_i to any $s_j \in S$ by Claim 5. There is no edge added from any $s_i \in S$ to an X_j since each X_i is adjacent to exactly one s_p, and these edges are in G^* (before applying the edits E') by Claims 2 and 4. And finally, there cannot be any edges added from some s_i to some s_j for $i \neq j$ since all such edges exist in G^*. □

Claim 7. If E' is an edge edit set of G^* such that the modified graph G' is quasi-threshold with $|E'| = k$, then we can find a collection $\mathcal{C}'$ of t 3-sets which is an exact cover of S.

Proof.

Recall that $S = \{s_1, s_2, \ldots, s_n\}$ and $\mathcal{C} = \{S_1, S_2, S_3, \ldots, S_m\}$. The proof of Claim 4 shows that the edited graph had at least $3(m-t)r$ deletions from S to $\bigcup X_i$ and so at most nr edges remain of the form sx with $s \in S$ and $x \in \bigcup X_i$.

We must now describe the other (at least) $k - 3(m-t)r = r - 3t$ deletions. Claim 3 deletes every $s_i s_j$ when s_i is adjacent to some $x_p \in X_p$ and s_j is adjacent to some $x_q \in X_q$ with $p \neq q$. Since S induces r edges in the original graph G^*, and we have (at least) $r - 3t$ other deletions, we have that S induces (at most) $3t$ edges in the edited graph G', and this maximum is obtained only when S induces a disjoint union of t triangles in G' such that each triangle is a triplet of vertices of S each belonging to one set in $\mathcal{C}$. This partition provides a subset of X_i sets which cover each s_j exactly once, giving our exact cover. □

This completes the proof of Theorem 2.9, as we have provided a polynomial time reduction from Exact 3-Cover to Quasi-Threshold Editing. □

2.5.1 Algorithms for Familial Groups

From the finite forbidden induced subgraph characterization of quasi-threshold graphs, the problem of modifying a graph to a closest quasi-threshold graph is *fixed-parameter tractable* when using either edge additions or deletions or both [19]. The trivial algorithm for quasi-threshold editing considers all possibilities of adding/deleting an edge between each pair of vertices in a forbidden P_4 or C_4, and so finding a closest quasi threshold graph with k edits runs in $O^*(6^k)$-time, where the notation $O^*(f(n,k))$ means $O(f(n,k)p(k,n))$ for some polynomial p.

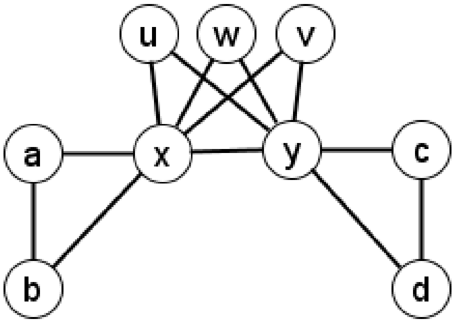


Figure 2.10: A graph which is 2 edge edits away from a quasi-threshold graph, with no single edge edit which reduces the total number of forbidden subgraphs.

The similar problem of modifying a graph to a quasi-threshold graph using only edge deletions is throughly discussed in Chapter 5.

For computational feasibility, we combined the above bounded search tree method with greedy edge-edit choices according to the measure of counting the total number of induced P_4s and C_4s in the graph. By testing every possible edge-addition and every possible edge-deletion, we (greedily) chose the edge edit that resulted in the largest improvement (that is, the largest decrease) in the total number of induced P_4s plus the number of induced C_4s in the graph. Greedy choices were made until the brute-force exact algorithm was able to execute on the modified graph within reasonable time.

We note here that only using greedy choices as described above may result in a process that cycles without reaching a quasi-threshold graph. In Figure 2.10, there are four P_4s with an endpoint in $\{a, b\}$ and the other endpoint in $\{c, d\}$. This graph is not quasi-threshold, and can be turned into a quasi-threshold graph by deleting the two edges ax and bx. However, any possible single edge edit will increase the total number of obstructions, and so the above greedy method would choose an edge edit with 0 net-gain over one of the edges ax and bx as removing ax or bx would create more P_4s than it would destroy (deleting ax would destroy two P4s but introduce 4 new ones: $abx\{u, v, w, y\}$, where $abxS$ denotes the set $abxs$ for each $s \in S$. For completion, we report the net gain of the remaining edge edits: bx, cy, dy all have net improvement of -2 for symmetric reasons to the ax removal. A deletion of an edge joining $\{u, v, w\}$ to $\{x, y\}$ has score 0, as does adding an edge inside $\{u, v, w\}$. Adding an edge from any of $\{u, v, w\}$ to any of

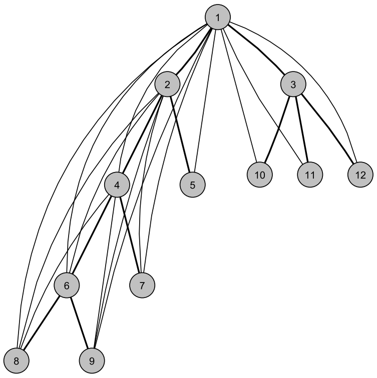


Figure 2.11: The degree of an actor does not determine its social rank.

$\{a, b, c, d\}$ will only create P4s without destroying any, so they also have negative improvement scores. Deleting xy would destroy the 4 original P_4s but would introduce 12 new ones. Adding an edge of the form $\{a, b\}y$ or $\{c, d\}x$ causes a net gain of +3 obstructions. Adding an edge from $\{a, b\}$ to $\{c, d\}$ destroys just one P_4 but adds 9 P_4s, along with a C_4. This shows that an edit (such as removing ab) with net improvement of 0 would be chosen by the greedy process, and the edit can be toggled repeatedly. If we restrict ourselves from undoing an edge edit, the result can still be poor due to deleting a large number of edges of 0 score without making progress (such as deleting an edge of a trivial 2-vertex component).

Observe also that the discussion in the previous paragraph shows that Figure 2.10 also serves as an example that a similar cograph-editing greedy process may suffer from the same non-termination. We stress, though, that our implementation utilized the greedy process only while significantly large improvements were being made, and that the brute-force (optimal) search replaced the greedy process once its computation was feasible.

In the next section, we analyze a selection of social networks by computing an approximate closest quasi-threshold graph with this combined search and greedy heuristic method.

2.5.2 Intra-communal Ranking

The importance of individuals to a network or a subnetwork is often measured by means of various vertex centrality metrics. These range from simple local properties such as vertex degree to global properties such as betweenness centrality.

The actors in a connected component of a quasi-threshold network naturally arrange themselves in a rooted tree representation. This correspondence can be used to extract an importance measure of each actor within the community. Intuitively, the root or top-most vertex of a familial group is the most important node and the others are ranked by virtue of the fact that each node can be regarded as the root of a subtree. The size of a subtree under an individual will be the relevant measure of importance here, rather than a metric such as vertex degree.

For instance, in the quasi-threshold community in Figure 2.11, vertex 6 has degree 5 and "oversees" 2 others, while vertex 3 has a lower degree of 4 but oversees 3 others. We perceive vertex 3 to have a more important role than vertex 6 in this community.

Hence, we define the *intra-communal rank* of a vertex v in a quasi-threshold community to be the number of vertices beneath v in the corresponding comparability tree. In the case that M vertices are structurally equivalent within the community in such a way that these M vertices all oversee d vertices beneath them in any associated comparability tree, then these M vertices can be given an intra-communal importance score of $d + \frac{M-1}{2}$.

This quantitative measure of intra-communal rank is merely one way to assign a value to a vertex that captures how important it is in its familial group. There are many possible ways such a measure could be defined, and an appropriate quantitative function is perhaps a topic for future research.

2.6 Case Studies

We present some example networks from the literature and the implied communities from a close quasi-threshold graph we computed.

2.6.1 Zachary's Karate Club

Zachary [150] studied the social relationships between individuals in a university karate club. The club suffered a division which split the club into two, and it was observed that the split very closely corresponded to a min-cut that separates the two opposing individuals of largest influence. A minimum cut (G, s, t) is a smallest set of edges of G such that the removal of this edge set will leave nodes s and t in different components.

The method of [62] for hierarchical clustering predicts roughly the same partition that Zachary observed after the karate club experienced its social fission, with the exception of vertex 3 being misclassified. The familial

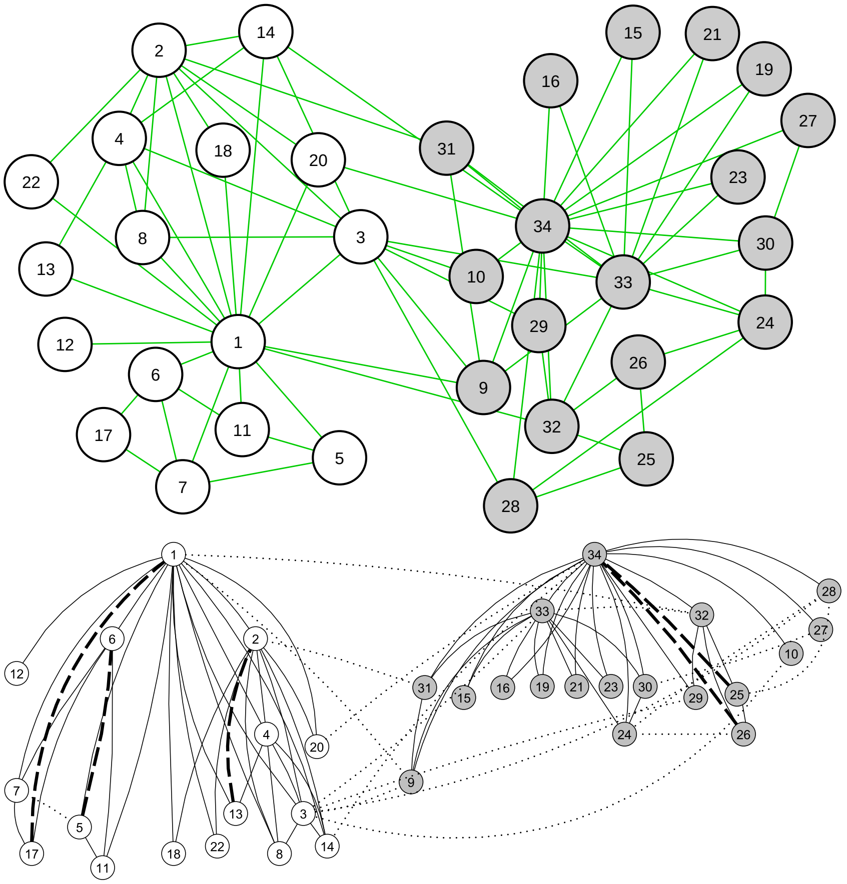


Figure 2.12: (top) Zachary's karate club network; (bottom) the quasi-threshold graph obtained after 21 edits.

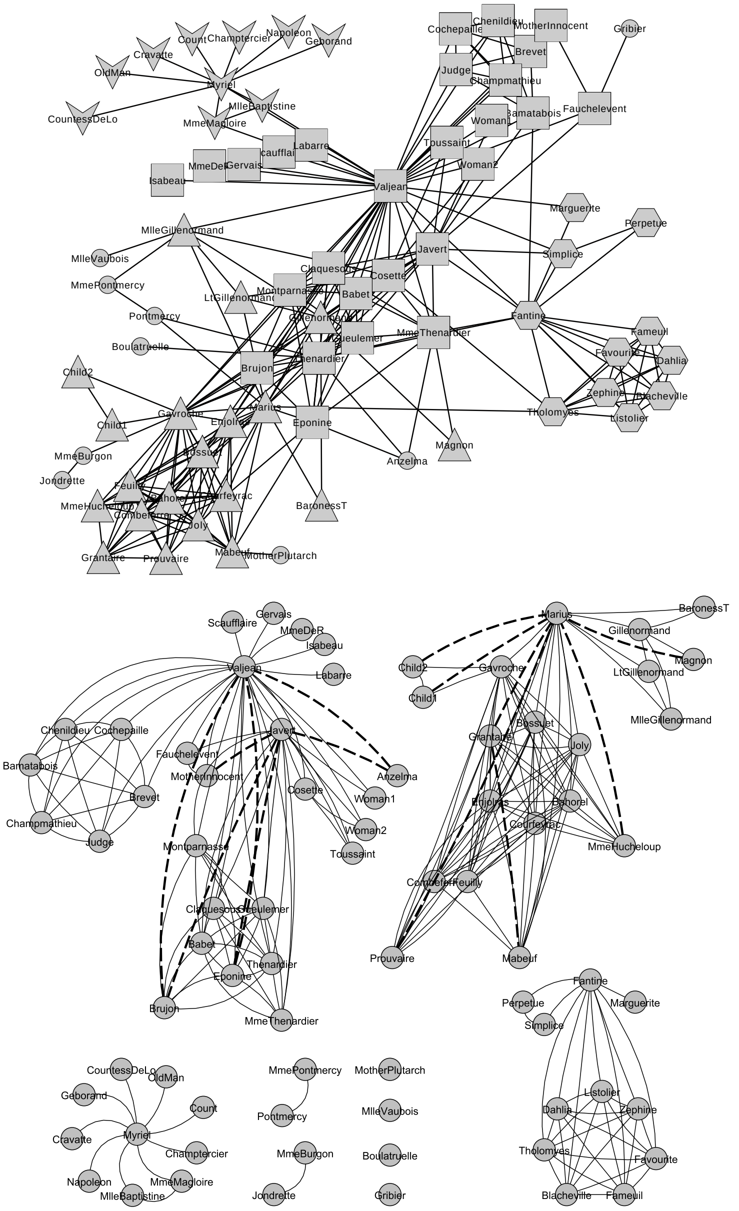


Figure 2.13: (top) Characters in the novel *Les Misérables.* The network is drawn in Cytoscape's spring embedding, while the approximated familial groups are distinguished by vertex shape and shading; (bottom) The familial groups found from the top network. The bold dashed lines represent edge additions, and deleted connections are not shown.

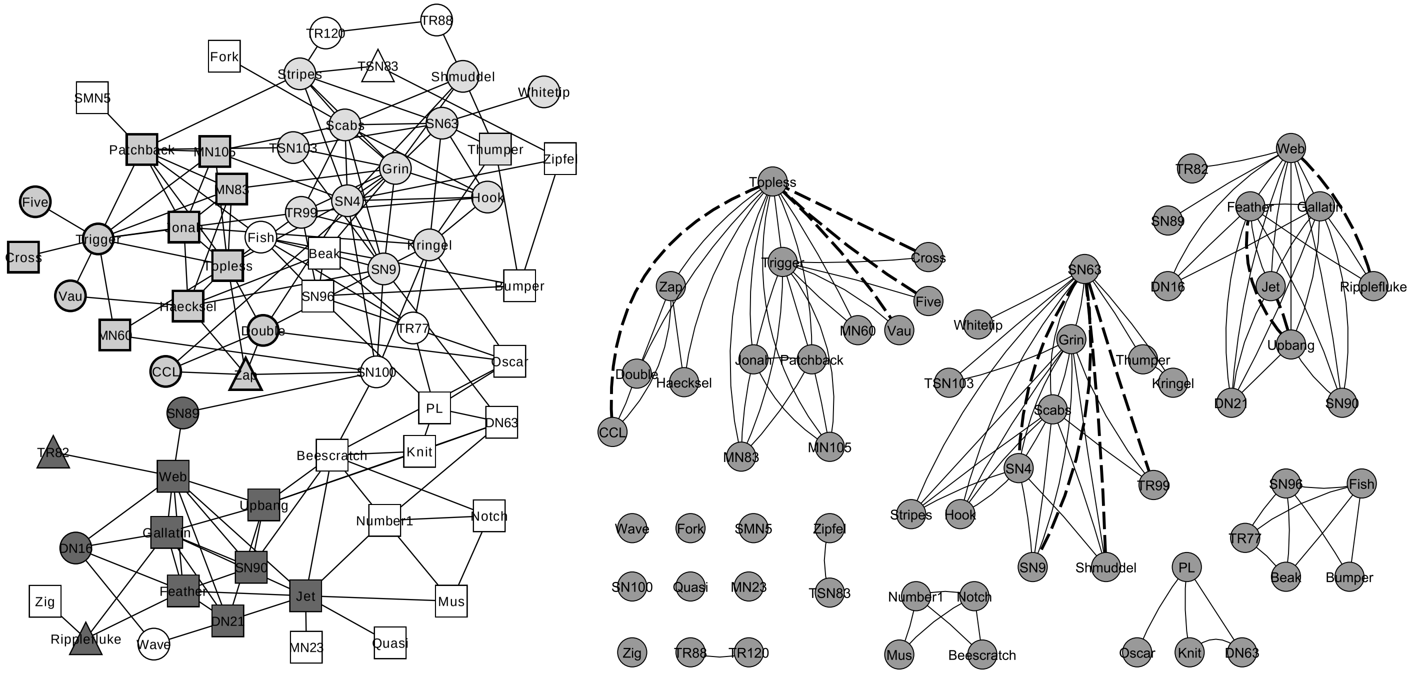


Figure 2.14: (left) Network of dolphin associations. Females are circle-shaped, males are square-shapes, and three individuals of unknown gender are depicted as triangles. The three main familial groups found are shown with light shading, light shading with thick outline, and dark shading; (right) The corresponding comparability trees of the quasi-threshold communities found in the dolphin network.

groups of the karate network identified by our approach are depicted in Figure 2.12 (top), where dashed lines represent edges from the network that were deleted and bold dashed lines represent new edges added in order to find the closest quasi-threshold graph. The obtained quasi-threshold partition groups the network into two groups, equivalent to the first two groups produced by the Girvan-Newman method. This network shows 21[5] edits, and this is optimal[6].

An interesting result of the tree structures revealed by our approach for the two groups is that it predicts exactly two distinct components with roots of vertices 1 and 34, while Zachary's method had begun with knowing that 1 and 34 are the conflicting leaders and found a minimum cut that separated 1 and 34. Sub-communities of the two major communities can be identified as subtrees of the quasi-threshold tree. Consider the removal of vertex 1: this leaves subtrees of $\{12\}$, $\{5, 6, 7, 11, 17\}$, $\{2, 3, 4, 8, 13, 14, 18, 20, 22\}$, which imply overlapping subcommunities when vertex 1 is regarded as a member of each of these subcommunities. We observe the similarity in the results implied by the dendrogram of Girvan and Newman, identifying a second-level community of $\{5, 6, 7, 11, 17\}$ as well, and vertex 12 quickly being separated from the remaining network. The larger of these three further decomposes into overlapping subcommunities when looking under vertex 2, and these communities are $\{1, 2, 18\}$, $\{1, 2, 22\}$, $\{1, 2, 20\}$, and $\{1, 2, 3, 4, 8, 13, 14\}$.

2.6.2 Communities in the Les Misérables Network and Character Importance

Les Misérables is a 19th-century novel by Victor Hugo containing 5 parts (or volumes) broken into 70 chapters. A network of 77 major and minor characters in the novel was constructed in [85] by joining two individuals with an edge if they exist in a chapter together.

Figure 2.13 shows the *Les Misérables* network and the computed familial groups. The familial groups found are distinguished by node shape and shading. The quasi-threshold graph obtained, after 64 edge edits, consists of three large nontrivial components and several smaller ones. The predicted leaders (roots) of these three components *Jean Valjean, Marius Pontmercy* and *Fantine* with implied intra-communal scores 27, 19, 10 (respectively), are key characters in the novel as is witnessed by the fact that their names

[5] We would like to thank Yarko Senyuta for noticing the error published in [110] where only 20 edits are reported.

[6] We thank Yarko Senyuta for verifying the optimality of this solution by showing that no solution exists in a bounded search tree of depth 20.

are titles to 3 of the 5 volumes. This quasi-threshold graph correctly isolates only minor characters into trivial groups.

2.6.3 Lusseau's Dolphin Network

Lusseau [101] studied a population of dolphins over a period of 7 years, building the social network depicted in the left-side network of Figure 2.14 by joining an edge between two dolphins if they were observed together significantly more often than was statistically expected. The community structure of this network was studied in [113], where the main community was identified as predominantly female and the male community split into two upon a temporary disappearance of several individuals.

Using 75 edge edits, our closest quasi-threshold graph found for the dolphin network is shown in the right-side of Figure 2.14. Our familial grouping supports the observed communities: it shows three main groupings, one of which is almost entirely female while each of the other two are mostly male. The remaining 15 dolphins are shown in the network as white nodes.

The number of small or trivial components in the edited dolphin network seems to be uncharacteristic of these edited networks, which may be concerning at first glance. But according to Lusseau[7], these bottlenose dolphins do not form hierarchies the same way as other social systems.

2.6.4 Grassland Species

The top network of Figure 2.15 is a network of grassland species interactions built in [37], and its hierarchical community structure was analyzed in [27]. The network contains 1007 induced obstructions (P_4s or C_4s) and we produce a quasi-threshold graph that is 34 edge edits away from it, depicted on the bottom of Figure 2.15. Each node corresponds to a type of organism such as plants (circle-shaped nodes), plant-eating organisms (square-shaped nodes) and parasitic organisms (the rest of the nodes.)

Interestingly, the root node of every non-trivial familial group was found to be a herbivore. It was found in [27] that several sets of parasites were grouped together not because they fed on each other but instead because they all fed on the same herbivore. Our familial groups strongly show that the herbivores play central roles in the organization of these species.

[7] Personal communication, May 30, 2013. "Socially though we do not have hierarchy formation like you might have in other types of social systems. The dispersal processes are also a bit different to what you might expect in our (human) notions of familial groups..."

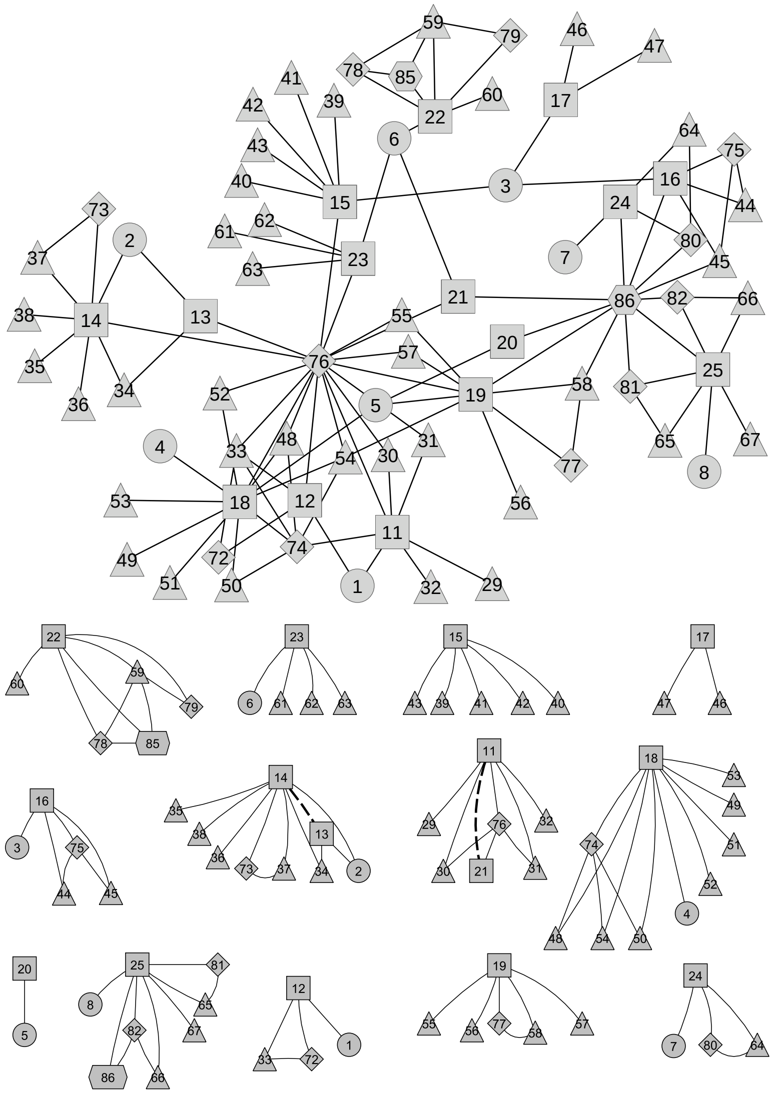


Figure 2.15: (top) Network of grassland species. Node categories are: plant(circle), herbivore(square), parasitoid(triangle), hyper-parasitoid(diamond), and hyper-hyper-parasitoid(hexagon). (bottom) The corresponding familial groups found after 34 edge edits. Bold dashed lines are edge additions and deleted edges are not shown.

2.6.5 College Football Network

[62] gives a network joining two American college teams together if they played against each other during the year 2000 football season. Evans writes that the data is likely the 2001 season [45], and corrects some of the conference assignments in the data[8]. In that football season, the 115 teams are grouped into 11 conferences, with a 12th group of independent teams. Teams are usually matched against their conference-mates, an average of about 7 games against teams within their own conference and 4 games outside of conference. Girvan and Newman extracted the community structure of this network and found a near-match to the expected partitions defined by conferences.

The network began with 613 edges, and our greedy method made 255 edge edits on the network to arrive at a (P_4, C_4)-free graph.

Quasi-threshold editing found exactly 12 connected components, almost perfectly matching the 12 conference groups as labeled by Evans. A table of the familial groups found is given in Table 2.1. The numeric groupings in the table correspond to the connected components found. The left and right icons in each row describe which conference that team is assigned to as given by the Newman dataset and the Evans dataset, respectively.

Figure 2.17 illustrates the intra-communal ranking of the teams in group 6, according to the discovered structure. The large score of Akron (the root) suggests that group 6 corresponds to the conference containing Akron, which is the *Mid American* conference. The relatively-high score of Buffalo is a strong suggestion that Buffalo belongs to the same conference as Akron. The very low scores of 0 for Central Florida and Connecticut tell us that although the structure of the scheduling that year seems to associate those two teams with the *Mid American* conference, these associations are very weak, even weaker than the ties for the other leaf-node teams of larger depth. The four teams ranked with 8.5 (Toledo, West Michigan, Miami Ohio, Central Michigan) were found to be equivalently structured in the community and so the placement order of those 4 teams in the comparability tree is arbitrary amongst each other. Similarly: Marshall, Ohio and Kent were found to be structurally equivalent, as were Northern Illinois, East Michigan and Ball State.

To illustrate how the scores are determined, Buffalo scores 12 because there are exactly 12 nodes below it in this tree (and in every tree obtained by permuting the order of any of the structurally-equivalent nodes). Toledo,

[8] We thank one of the referees for bringing to our attention the corrected conference assignment made by Evans.

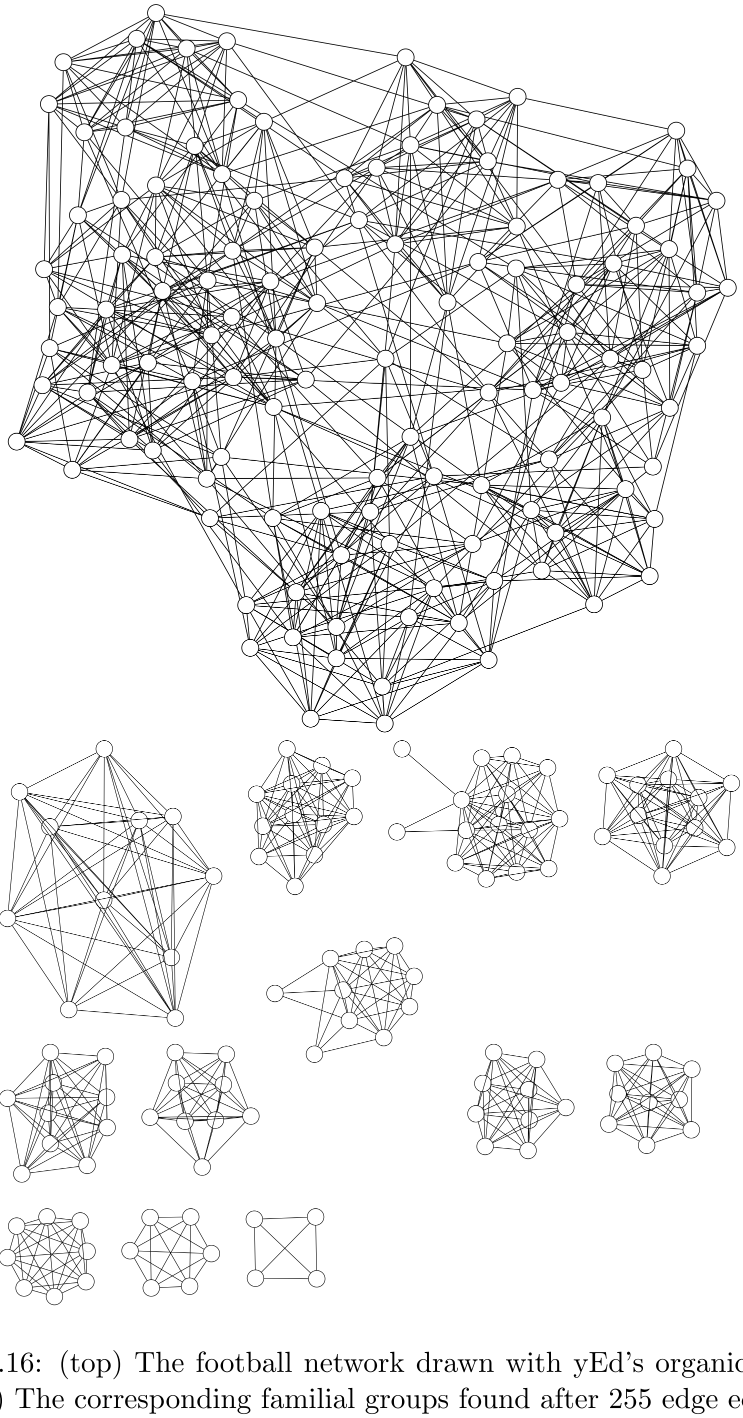


Figure 2.16: (top) The football network drawn with yEd's organic layout; (bottom) The corresponding familial groups found after 255 edge edits. Interestingly, exactly 12 components were found, mostly corresponding to the 12 conferences that partitioned the teams.

Familial Groups Found in the Football Network		
1	∘ ∘	Clemson
	∘ ∘	Wake Forest
	∘ ∘	Maryland
	∘ ∘	North Carolina State
	∘ ∘	Florida State
	∘ ∘	Virginia
	∘ ∘	Georgia Tech
	∘ ∘	Duke
	∘ ∘	North Carolina
2	• •	Miami Florida
	• •	Virginia Tech
	• •	Boston College
	• •	West Virginia
	• •	Syracuse
	• •	Pittsburgh
	• •	Temple
	• •	Rutgers
	■ ■	Navy
	■ ■	Notre Dame
3	▷ ▷	Michigan State
	▷ ▷	Indiana
	▷ ▷	Northwestern
	▷ ▷	Wisconsin
	▷ ▷	Michigan
	▷ ▷	Iowa
	▷ ▷	Purdue
	▷ ▷	Ohio State
	▷ ▷	Minnesota
	▷ ▷	Illinois
	▷ ▷	Penn State
4	▶ ▶	Missouri
	▶ ▶	Oklahoma State
	▶ ▶	Baylor
	▶ ▶	Colorado
	▶ ▶	Kansas State
	▶ ▶	Kansas
	▶ ▶	Texas Tech
	▶ ▶	Iowa State
	▶ ▶	Nebraska
	▶ ▶	Texas A&M
	▶ ▶	Oklahoma
	▶ ▶	Texas
5	□ □	Alabama Birmingham
	□ □	East Carolina
	□ □	Houston
	□ □	Louisville
	□ □	Memphis
	□ □	Southern Mississippi
	□ □	Tulane
	□ □	Army
	□ □	Cincinnati
6	◇ ◇	Marshall
	◇ ◇	Northern Illinois
	◇ ◇	Western Michigan
	◇ ◇	Akron
	◇ ◇	Ball State
	◇ ◇	Bowling Green State
	◇ ◇	Buffalo
	◇ ◇	Central Michigan
	◇ ◇	East Michigan
	◇ ◇	Kent
	◇ ◇	Miami Ohio
	◇ ◇	Ohio
	◇ ◇	Toledo
	■ ■	Central Florida
	■ ■	Connecticut
7	⋆ ⋆	Oregon State
	⋆ ⋆	Arizona State
	⋆ ⋆	California
	⋆ ⋆	UCLA
	⋆ ⋆	Arizona
	⋆ ⋆	Washington
	⋆ ⋆	Oregon
	⋆ ⋆	Stanford
	⋆ ⋆	Washington State
	⋆ ⋆	Southern California
8	♣ ♣	Nevada Las Vegas
	♣ ♣	San Diego State
	♣ ♣	Wyoming
	♣ ♣	Brigham Young
	♣ ♣	Utah
	♣ ♣	Colorado State
	♣ ♣	New Mexico
	♣ ♣	Air Force
9	◁ ⊕	North Texas
	■ ⊕	Utah State
	◁ ⊕	Arkansas State
	◀ ⊕	Boise State
	◁ ⊕	Idaho
	◁ ⊕	New Mexico State
10	♡ ♡	Arkansas
	♡ ♡	Auburn
	♡ ♡	Alabama
	♡ ♡	Florida
	♡ ♡	Kentucky
	♡ ♡	Vanderbilt
	♡ ♡	Mississippi State
	♡ ♡	South Carolina
	♡ ♡	Tennessee
	♡ ♡	Mississippi
	♡ ♡	Georgia
	♡ ♡	Louisiana State
11	◀ ◀	Hawaii
	□ ◀	Texas Christian
	◀ ◀	Fresno State
	◀ ◀	Rice
	◀ ◀	Southern Methodist
	◀ ◀	Nevada
	◀ ◀	San Jose State
	◀ ◀	Texas El Paso
	◀ ◀	Tulsa
12	◀ ■	Louisiana Tech
	◁ ■	Louisiana Monroe
	◁ ■	Middle Tennessee State
	◁ ■	Louisiana Lafayette

Table 2.1: The 12 connected components found after greedily editing-out the P_4s and C_4s from the football network. The left symbol is the conference assignment as given in Girvan and Newman's dataset, while the right symbol corresponds to Evans' corrected conference assignment. The grouping found corresponds almost exactly to the 12 conference groups described by Evans. The conference labels are depicted by the legend below.

∘ Atlantic Coast	□ Conference USA	⋆ Pac 10	• Big East
■ IA Independents	♡ SEC	▷ Big 10	◇ Mid American
◁ Sunbelt	▶ Big 12	♣ Mountain West	◀ Western Athletic
⊕ Big West			

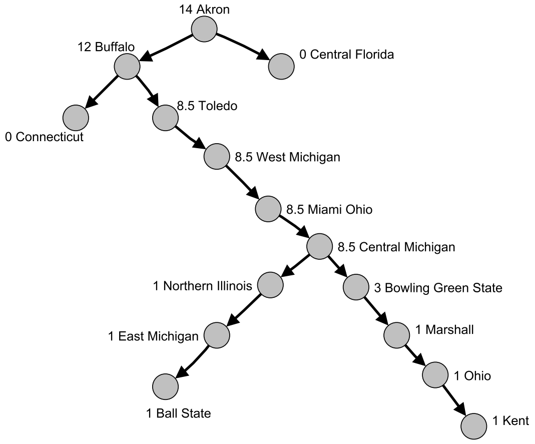


Figure 2.17: The implied comparability tree corresponding to one particular familial group. The intra-communal ranking is also given for each team. This is group 6 in Table 2.1.

however, is in an equivalence group {Toledo, West Michigan, Miami Ohio, Central Michigan}, and this group directly oversees 7 nodes below them. The intra-communal score of $d + \frac{M-1}{2}$ with $d = 7$ and $M = 4$ gives each of these 4 teams a score of 8.5.

2.7 Summary

The main contribution of this chapter is a new definition for community structure (familial groups) based on the graph-theoretical concept of forbidding small induced subgraphs. We give evidence that editing-out P_4s and C_4s in order to obtain a quasi-threshold graph yields meaningful clusterings in real networks.

We show that the computational problem of edge-editing to a nearest quasi-threshold graph is NP-complete, resolving the open complexity status of this problem as mentioned in [18], [102] and in [97].

Familial groups are a natural and meaningful relaxation of many standard structures widely-used in social community definitions such as cliques and k-plexes and their generalizations. They are robust against the removal of network nodes, a characterization that is not guaranteed in most other definitions of a social community. This robustness is in the sense that if the unique leader of a group is removed, the remaining individuals will either still form a familial group with a leader or will decompose into several familial groups, each with a respective leader.

Familial groups automatically provide a ranking of the individuals within a group and a hierarchical arrangement of a group itself - a unique feature that, to our best knowledge, no other existing model of community structure provides. These communities also retain many of the properties that a highly-connected group should have such as having a low diameter. In fact, the diameter of a familial group is at most 2, since the editing process ensures a universal vertex in each connected connected.

The notion of familial groups presented here can easily be modified to handle more network information, including weighted nodes and edges, directed edges, and weighted edge-edit operations. The following chapter will explore a possible way to incorporate directed and weighted edges into the theory of familial groups.

Chapter 3

Familial Groups for Hierarchical Organization

3.1 Historical Perspective

In 1994, Krackhardt [88] provided definitions for a purely hierarchical organization in which there is one "boss" who oversees any number of subordinates, each of which possibly overseeing their own set of subordinates. Krackhardt represented these structures as directed rooted trees, where each node is directed toward each of its children. He observed that these "barebones" trees are perhaps too "fragile" and called extra edges "redundant" when they pointed from a boss to a subordinate's subordinate. Krackhardt gave four defining properties of this structure which were then used to measure how close a given directed network was to being perfectly hierarchically organized.

Recently, Everett and Krackhardt [46] noted that the four defining properties were not completely independent from each other, and proposed modifications to the scoring system to measure when a given network exhibits the desired or expected properties of a hierarchical organization.

In 2002, independently from the works above, attempts to measure hierarchical organizations in networks came in from Ravasz and Barabási and others [42] [1] [123] where the amount of organization in a network was hypothesized to be measurable through a density measure called the *clustering coefficient.*

In 2007, Sales-Pardo et al. [126] comment that checking for desired (and seemingly irrelevant) properties like the clustering coefficient does not yield an "unsupervised" method to extract these hierarchical organizations. They also note that hierarchical clustering methods lack in any "sound general criterion to determine the relevant levels on the hierarchy."

We introduced *familial groups* [109] in Section 2.4 whose definition evolved from imposing certain structural properties in social communities. An immediate benefit of the structure of familial groups was that the individuals in identified communities were found to naturally arrange themselves in a hierarchical organization.

In this chapter, we show that familial groups can be used to define and characterize the structure of social hierarchy in networks. Furthermore, we extend the definitions of familial groups to handle directed and weighted networks.

3.2 Graph-theoretic Framework for Hierarchical Organization

We say that vertex u *points to* vertex v is there is a directed edge from u to v, and we write (u, v) or uv when the context is clear. An *out-tree* [88] or an *arborescence* [136] [46] [40] is a directed rooted tree where every node points to each of its children. It represents a hierarchical organization of nodes where an individual node can - for example - pass orders or influence along its directed edges (that is, to each of its children.) One can interpret this hierarchical organization as a boss and his/her subordinates.

It makes sense, then, that an edge can be directed from a boss to its subordinate's subordinate since demands can typically be passed down a chain of authority. Krackhardt [88] called these edges *redundant*[9] and excluded these edges from the study of hierarchical structures. Redundancy, however, serves as an integrity-check in that if an individual x is directed toward all of its subordinates and their descendants, it reinforces the hypothesis that x is the root node of the hierarchical organization.

Consider an arborescence T with root r. The *comparability graph of an arborescence* [40], $G(T)$, is a directed graph on the same node set as T in which (u, v) is an arc in $G(T)$ when v is a descendant of u. In particular, the root of such a tree is adjacent to and points toward every other node in the tree. $G(T)$ is the *transitive closure* of the arborescence T, meaning that whenever there is a directed path from x to y in T, there is a directed edge from x to y in $G(T)$.

The *underlying graph* of a directed graph D is the graph resulting from D when the orientation of each edge is ignored. The class of graphs which are a comparability graph of an arborescence is exactly the class of quasi-threshold graphs (defined in Section 1.3.2).

[9] quoting: "More lines than [a tree] creates multiple paths and cycles between points. In a sense, these multiple paths are redundant in graph-theory terms, and they disrupt the stoic, bare-bones nature of the pure out-tree structure.

Theorem 3.1. *[145] A connected graph is a comparability graph of an arborescence if and only if it has no induced subgraph isomorphic to a path on four vertices (a P_4) or a cycle on four vertices (a C_4).*

3.3 Hierarchical Organization of Individuals in a Network

Krackhardt [88] gave four characterizing properties of out-trees:

i) The digraph is weakly connected (that is, this condition only requires connectivity on the underlying graph).

ii) The digraph is graph hierarchic, meaning that if u can reach v with a directed path, then v cannot reach u.

iii) The digraph is graph efficient, meaning that it has no redundant edges.

iv) Every pair of vertices has a least upper bound, that is, a closest boss who has authority over both of them.

For each of these four criteria, he gave a real-valued score function from 0 to 1 for any given network, where 0 represents a high degree of violation of that criteria and 1 representing the expected structure of an out-tree. We elaborate on the score function for each criteria below. The number of nodes in a given digraph is n.

i) $1-\frac{t}{\binom{n}{2}}$, where t is the number of pairs of vertices which are unreachable from each other. This corresponds to a score of 0 if the digraph is an independent set and 1 if it is weakly connected.

ii) If the strongly-connected components of the digraph are $C_1, \ldots, C_t$ then this function is $1 - \frac{\sum_{i=1}^{t} \binom{|C_i|}{2}}{\binom{n}{2}}$. That is, every pair of vertices which is in a strongly connected component counts as a violation, summed over all possible pairs. This function is 0 if the entire digraph is strongly connected and 1 if it acyclic.

iii) Say that a connected component C_i in the underlying graph has n_i nodes. Then a tree structure would expect $n_i - 1$ edges. Let m_i be the number of edges of C_i. Then the i^{th} component has $m_i - (n_i - 1)$ violating edges. The function then is $1 - \frac{\sum_i (m_i - n_i + 1)}{\sum_i (\binom{n_i}{2} - (n_i - 1))}$. This function evaluates to 0 if the underlying graph is a clique, and to 1 if the underlying graph is a forest (i.e. a disjoint collection of trees.)

iv) A violating pair is a pair of nodes, u and v, in the same weakly-connected component such that there is no x for which there is a directed path from x to u and x to v. The least upper bound x of u and v is allowed to be either u or v as well. The score function is $1 - \frac{t}{z}$ where t is the number of violating pairs and z is the total number of non-adjacent pairs in the same weakly connected component. That is, for each weakly connected component C with n_C nodes and m_C edges, $z = \binom{n_C}{2} - m_C$. This function is 0 when every pair of nonadjacent nodes have no directed path joining them. For instance, if every node is a source or a sink, there are no directed paths of length 2 and this function evaluates to 0. It evaluates to 1 when, for instance, there is a vertex which has a directed path to all other nodes in the weakly connected component, although this is not the only manner in which it can score 1.

Krackhardt measured these parameters in Erdős-Rényi graphs (see Section 4.4) and observed that the four measures seemed to independently be valued high (~ 1) or low (~ 0).

Later, Everett and Krackhardt [46] showed that these four dimensions of hierarchical organization are not independent of each other. Specifically, they showed that condition (ii) followed from the other three conditions. Weaker, but similar, statements of the four condition were proposed as replacement. They also propose that in certain applications, for instance, where two-way arcs can exist, that only a certain triplet of the four conditions be used.

The 1994 paper [88] concluded with a cautionary note that these measures are very sensitive to density and that data-collecting methods may be prone to including many redundant edges, inadvertently reducing the measures of hierarchical organization. This suggests that a method of analysis which filters-out, or at least embraces, redundancy could prove to be more fruitful in measuring organizational structure.

For any directed graph $G = (V, E)$, the *transitive closure of G* is the graph $H = (V, E \cup E^+)$ where a directed edge (u, v) exists in E^+ if there is a directed path from u to v in G. Adding all redundant edges, then, is akin to obtaining a transitive closure. The following observation is immediate:

Observation 3.2. The transitive closure of an out-tree is a comparability graph of an arborescence.

In general, the advantage of adding redundancy to a model is to reinforce the initial data and to further support any conclusions or predictions made by the model. We believe the same holds in this case of hierarchical organization in networks.

3.4 Familial Groups in Directed Networks

Using the method of familial groups in Section 2.4, we could attempt to extract hierarchical organization in a directed network by ignoring edge directions, editing to a closest quasi-threshold graph, and then arranging the individuals in the implicated tree structure and directing edges downward. But when ignoring initial edge directions, this could result in a hierarchical structure that is invalid or very inaccurate.

Since a comparability graph of an arborescence is a directed structure (having a quasi-threshold underlying graph), instead of restricting the local structure by P_4s and C_4s, we can impose restrictions on the local directed structure and define an edit distance with respect to this.

There are some situations in which it makes sense that between any two nodes u and v, only one directed edge can exist between them. That is, if u points to v, then v cannot possibly point to u. For example, if u and v exist in an extended family, u might be a descendant of v or v a descendant of u (or neither), but it is impossible for both to occur. A single directed edge can be drawn between them (ancestor pointing to descendant) and it would be meaningless to have two individuals pointing to each other.

On the other hand, a trust network in which a directed edge (u, v) exists if v follows advice from u may very well contain edges (u, v) and (v, u) simultaneously. This can be made into a simple underlying graph by instead joining directed edge (u, v) if v follows advice from u *more often than* u following v. In this case of following each others' advice equally, from their perspective, one is no higher than the other in the social hierarchy and so the edge between them can be removed or ignored and their placing in the hierarchy will de determined by the other actors in the network.

Again, depending on what desired property is being captured in the network, ignoring the edge might not be an ideal approach, since u and v following each other's advice ten times per week may be more socially significant than u following the advice of w once. But in these cases where cooperation or communication channels are of primary interest, the underlying graph structure may be enough to maintain and analyze.

With all these possibilities and considerations, we emphasize that the

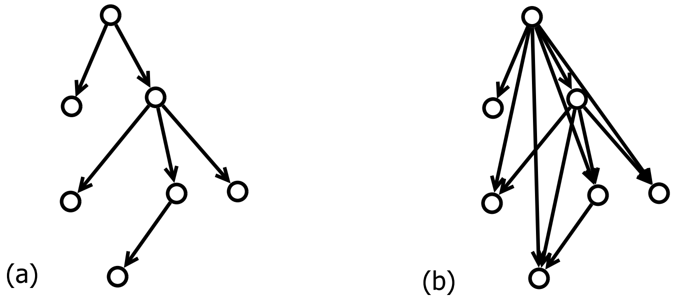


Figure 3.1: (left) An out-tree as used by Krackhardt [88]; (right) the transitive closure of the left out-tree.

network model is dependent on the application it is modeling and the property that the network analyst is interested in. We elaborate on some of these network models in the following sections.

3.4.1 Directed Networks with a Simple Underlying Graph

In an application where at most one directed edge can exist between a pair of nodes, the familial group can be defined exactly as a comparability tree of an arborescence, with edges pointing away from the root vertex and towards to leaves. Given a directed network, an edge-edit could be:

i) the deletion of a directed edge of cost α

ii) the creation of a directed edge of cost β

iii) the reversal of a directed edge of cost γ

While a reversal of an edge can be regarded as a deletion followed by a creation, it is not always appropriate to weigh a reversal edit as $\alpha + \beta$. We propose a default weighting of $\alpha = \beta = \gamma = 1$ unless the application in the model suggests otherwise. For instance, adding a redundant edge in order to complete the transitive graph structure may not necessarily be penalized as much as deleting a known connection, and so perhaps for those applications, adding an edge could have a very small (but positive) value $\epsilon > 0$ assigned to β.

Figure 3.1(a) depicts the ideal hierarchical structure as described in [88], and Figure 3.1(b) is the same out-tree shown in (a) but with all additional "redundant" edges included. Let us assign a name to this structure:

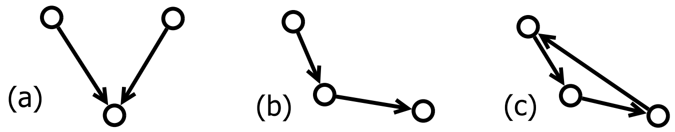


Figure 3.2: The forbidden triadic configurations for transitive out-forest when the underlying network must be simple.

Definition 3.3. A *transitive out-tree* is the transitive closure of an out-tree. A disjoint collection of transitive out-trees will be called a *transitive out-forest.*

Thus Figure 3.1(b) depicts a transitive out-tree. When data-collecting methods include measurements for almost all pairs of individuals, we expect many of these redundant edges to appear in the resulting network model. Thus we are interested in a closeness score to one of these transitive out-trees. Note that these directed structures are not exactly characterized as transitively-oriented quasi-threshold graphs, since reversing all the edges of Figure 3.1(b) would still be a transitively-oriented quasi-threshold graph while not having the rooted structure. We are particularly interested in an orientation that specifically directs edges away from the root and towards the tree leaves.

Analogous to how familial groups are defined for undirected graphs, we define a familial group of a directed simple graph G as a (weakly) connected component of a closest graph G' which is a disjoint collection of transitive out-trees.

Definition 3.4. Given a directed network G with simple underlying structure, the *simple directed familial groups* (or simply, the *familial groups*) of G is the set of transitive out-trees obtained from a closest transitive out-forest.

In order to edit a directed simple graph to a closest transitive out-forest, we can take a similar approach as in that for familial groups on undirected graphs and characterize the desired structure in terms of local structure. The transitive out-tree structure of a familial group, for instance, requires that if two vertices u_1 and u_2 point to v, then either u_1 points to u_2 or vice versa. The forbidden configurations of a transitive out-forest are depicted in Figure 3.2.

In the framework where an edge deletion costs α, an edge addition costs β, and an edge reversal costs γ, we can formulate the parameterized problem of finding simple directed familial groups of a directed simple graph:

Problem 5. Transitive Out-tree Editing(G, k):
Given a directed graph $G = (V, E)$, is there a set of x edge deletions, y edge additions and z edge reversals such that the resulting graph is a transitive out-forest and $\alpha x + \beta y + \gamma z \leq k$?

As mentioned earlier, a natural definition of this problem would set $\alpha = \beta = \gamma = 1$ unless another interpretation is suggested by the problem application.

Since the forbidden configurations are finite in number, we can immediately describe a fixed-parameter tractable algorithm to solve this.

As every pair of vertices has three possible states of having an arc between them (two directional arcs or no arc at all) there are 27 configurations of a triplet in total. Although three forbidden configurations are given in Figure 3.2, when symmetries are counted, there are three forbidden configurations of type (a), six forbidden configurations of type (b), two forbidden configurations of type (c), for a total of 11 forbidden triadic configurations. A brute-force algorithm would find a forbidden triplet and branch on replace their adjacencies with one of the 27-11=16 valid configurations and repeat.

Algorithm 4: Transitive out-forest editing algorithm.

```
Algorithm TransOutEdit($(G, k)$):
Input: A Graph $G = (V, E)$
Output: Yes if $G$ can be edited to a transitive out-forest with at
        most $k$ edits, and No otherwise.

while (there exists a forbidden triple of nodes) && ($k > 0$) do
    Branch on the possible ways of editing the triple into a valid
    triple as per Table 3.1, and reduce the parameter $k$ accordingly;
end
if $k \geq 0$ then
    return Yes;
end
return No;
```

The minimal branching rules are summarized in Table 3.1. Under the assumption that $\alpha = \beta = \gamma = 1$, Algorithm 4 is fixed parameter tractable and the branching factor for each of the three branching rules are found to be:

i) $T(k) = 4T(k-1) \rightarrow T(k) = O^*(4^k)$

ii) $T(k) = 3T(k-1) + 4T(k-2) \rightarrow T(k) = O^*(4^k)$

iii) $T(k) = 3T(k-1) + 3T(k-2) \rightarrow T(k) = O^*(3.792^k)$

Theorem 3.5. *Transitive out-tree editing using edge additions, deletions and reversals can be solved in $O^*(4^k)$.*

Configuration	Minimal Edge Edits	Parameter
$u \rightarrow v$ and $w \rightarrow v$	del (u, v)	$k \leftarrowtail k - \alpha$
	del (w, v)	$k \leftarrowtail k - \alpha$
	add (u, w)	$k \leftarrowtail k - \beta$
	add (w, u)	$k \leftarrowtail k - \beta$
$u \rightarrow v$ and $v \rightarrow w$	del (u, v)	$k \leftarrowtail k - \alpha$
	del (v, w)	$k \leftarrowtail k - \alpha$
	add (u, w)	$k \leftarrowtail k - \beta$
	rev (u, v), add (w, u)	$k \leftarrow k - \beta - \gamma$
	rev (v, w), add (w, u)	$k \leftarrow k - \beta - \gamma$
	rev (u, v), add (u, w)	$k \leftarrow k - \beta - \gamma$
	rev (v, w), add (u, w)	$k \leftarrow k - \beta - \gamma$
$u \rightarrow v$ and $v \rightarrow w$ and $w \rightarrow u$	del (u, v) and (v, w)	$k \leftarrow k - 2\alpha$
	del (u, v), (w, u)	$k \leftarrow k - 2\alpha$
	del (v, w), (w, u)	$k \leftarrow k - 2\alpha$
	rev (u, v)	$k \leftarrow k - \gamma$
	rev (v, w)	$k \leftarrow k - \gamma$
	rev (w, u)	$k \leftarrow k - \gamma$

Table 3.1: Branching rules for transitive out-tree editing with the addition, deletion, and reversal operations.

3.4.2 Transitive out-tree editing without reversal operations

Following the editing framework of [141] and [142] where the problem of editing a directed graphs to a transitive directed graph, we consider here the modified problem of editing with only the addition and deletion operations. Note that arc reversals can be simulated with an arc deletion followed by an arc addition in the opposite direction, and so this framework does not loose expressibility power in terms of the problems it can tackle. Editing directed graphs to make them transitive is equivalent to forbidding the configuration (b) in Figure 3.2. Weller et al. gave a $O^*(2.57^k)$-time algorithm for Transitivity Editing and $O^*(2^k)$-time algorithm for Transitivity Deletion.

Note that Algorithm 4 still solves the modified version of Transitive Out-tree Editing with only edge additions and deletions allowed, but we only have to specify a new set of minimal edits to branch on. We use $\alpha = \beta = 1$ in Table 3.2 to describe these edits.

Configuration	Minimal Edge Edits	Parameter
$u \to v$ and $w \to v$	del (u, v)	$k \leftarrowtail k - 1$
	del (w, v)	$k \leftarrowtail k - 1$
	add (u, w)	$k \leftarrowtail k - 1$
	add (w, u)	$k \leftarrowtail k - 1$
$u \to v$ and $v \to w$	del (u, v)	$k \leftarrowtail k - 1$
	del (v, w)	$k \leftarrowtail k - 1$
	add (u, w)	$k \leftarrowtail k - 1$
$u \to v$ and $v \to w$ and $w \to u$	del $(u, v), (v, w)$	$k \leftarrow k - 2$
	del $(u, v), (w, u)$	$k \leftarrow k - 2$
	del $(v, w), (w, u)$	$k \leftarrow k - 2$
	del (u, v), add (v, u)	$k \leftarrow k - 2$
	del (v, w), add (w, v)	$k \leftarrow k - 2$
	del (w, u), add (u, w)	$k \leftarrow k - 2$

Table 3.2: Branching rules for transitive out-tree editing using only the addition and deletion operations.

The number of nodes explored in a bounded search tree that uses these editing rules results in a number of nodes satisfying these recurrences:

i) $T(k) = 4T(k - 1) \to T(k) = O^*(4^k)$

ii) $T(k) = 3T(k - 1) \to T(k) = O^*(3^k)$

iii) $T(k) = 6T(k - 2) \to T(k) = O^*(2.450^k)$

We can also consider the related deletion-only problem:

Problem 6. Transitive Out-tree Deletion(G, k):
Given a directed graph $G = (V, E)$, is there a set of k edge deletions such that the resulting graph is a transitive out-forest?

The minimal deletion sets are summarized in Table 3.3.

Configuration	Minimal Edge Edits	Parameter
$u \to v$ and $w \to v$	del (u, v)	$k \leftarrowtail k - 1$
	del (w, v)	$k \leftarrowtail k - 1$
$u \to v$ and $v \to w$	del (u, v)	$k \leftarrowtail k - 1$
	del (v, w)	$k \leftarrowtail k - 1$
$u \to v$ and $v \to w$ and $w \to u$	del $(u, v), (v, w)$	$k \leftarrow k - 2$
	del $(u, v), (w, u)$	$k \leftarrow k - 2$
	del $(v, w), (w, u)$	$k \leftarrow k - 2$

Table 3.3: Branching rules for transitive out-tree edge-deletion.

Clearly, the first two branching rules create $O(2^k)$ nodes. The third branching rule results in a recurrence of $T(k) = 3T(k-2)$, with an effective branching factor of 1.733. Thus we have:

Theorem 3.6. Transitive Out-tree Deletion*(G, k) can be solved in* $O^*(2^k)$ *time.*

3.4.3 Weighted Directed Framework

In the case of weighted networks in which each edge or arc is associated with a score, we can still naturally define familial groups. Since edge weights are usually a measure of the strength of the edge (except, perhaps, in networks in which the weight is a distance measure) it should make sense then that deleting an edge should cost the weight of the edge[10].

In an undirected setting, familial groups would still be defined in terms of the forbidden P_4s and C_4s, and the weights would only be used in determining the costs of editing.

In the directed network setting in which data is expected to satisfy transitivity, we propose that arcs can be added with a minimal cost if it satisfies transitivity. As mentioned in an earlier section, these edges that observe transitivity are a form of data redundancy and if $u \to v$ and $v \to w$, then it may make sense to allow $u \to w$ for free, or perhaps a very small cost.

An example to motivate this definitions is depicted in Figures 3.3 and 3.4. We consider 5 NHL hockey teams of one division after having played an entire 48-game season. The league consists of 30 teams divided into 2 conferences and each conference divided into 3 divisions. The five teams played

[10] There are always exceptions to this model, of course. In a network in which edges are roads or pathways and weighed with the amount of traffic that passes it, deletion or blocking a road of low traffic or of high traffic could feasibly require the same amount of work.

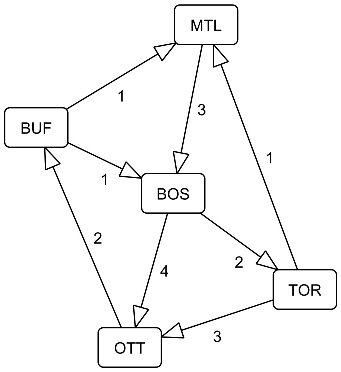


Figure 3.3: The teams from the North-East Division of the Eastern Conference of the 2012-2013 NHL season. Team x points to team y if x beat y throughout the regular season more often than y beat x. Each edge is weighted by the number of more wins x had over y when playing against each other.

all of their games within their conference that season. In Figure 3.3, we point u to v if u beat v more often than v beat u. The arcs are weighed by the number of wins u had over v minus the number of wins v had over u. If two teams, such as MTL and OTT, beat each other equally often, then an arc joining them would have score 0 and so it is not drawn.

In editing the network into a closest directed out-tree, we find that three deletions are required to break the directed cycles. Each of those deletions are of weight 1. There is a set of five arcs that must be added to satisfy transitivity to complete the out-tree structure, as shown in Figure 3.4. When arc additions are allowed to be added with small ϵ cost to complete transitivity requirements, this $3 + 5\epsilon$ edit set is the minimum possible solution. It turns out that this arrangement correctly predicts the final arrangement of these five teams after their 48 game season: MTL won the division with 63 points, BOS had 62 points, and TOR, OTT and BUF earned 57, 56 and 48 points, respectively.

There is still much work to be done in formalizing when arc additions can be added with minimal cost. Perhaps a framework could be defined in which deletions are made to only destroy the configurations (a) and (c) from Figure 3.2, and then a graph square or even transitive closure could be applied to add redundant arcs.

We believe the discussion in this thesis shows that there is definite merit and opportunity to apply the concept of quasi-threshold graphs and tran-

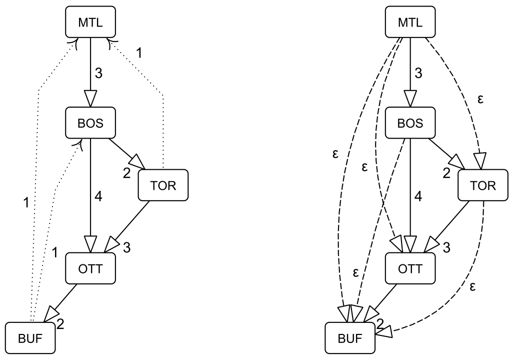


Figure 3.4: (Left) The three deletions (dotted lines) with total cost 3 to destroy transitivity obstructions. (Right) The addition of 5 edges (dashed) of consistent implication, each costing ϵ. The total weight of deletions and additions is $3 + 5\epsilon$.

sitive out-trees to weighted directed networks. A complete formalization would require an implementation and several test cases to justify the definitions and we leave this as a future work.

Chapter 4

Network Measures: Diameter and Distribution

As social networks are created to analyze a collection of relationships, a number of attributes of networks (or vertices or edges of a network) have been defined quantitatively for more meaningful and comparative analysis. We provide some of the fundamental definitions of social network attributes here.

The *centrality* of a vertex or an edge is a measure of how important that vertex or edge is to the network. This is, of course, vague and can be fathomed in many different ways, and so there are many types of centrality measures in regular use. Perhaps the most simplest notion of vertex centrality is the *degree centrality*, which simply assigns a centrality score to a vertex equal to its degree. The simplicity of this definition makes its use rather limited: in Figure 4.1, the two labeled vertices have the highest degree centrality measure but are structurally very different. If, for instance, these nodes are individuals and an edge between two nodes represents an email communication between them, then A could be regarded as some sort of spam machine which only communicates to seemingly unrelated nodes. On the other hand, B (with the same degree as A) communicates to a group of individuals who also communicate among each other. This sort of quality can be captured by counting the number of edges in the neighbourhood of a vertex, called the *clustering coefficient*.

Definition 4.1. Clustering Coefficient. In a graph $G = (V, E)$, the *clustering coefficient* C_v of a vertex v with open neighbourhood $N(v)$ of size at least 2 is the ratio of the number of edges in $N(v)$ and the number of vertex pairs in $N(v)$. That is,

$$C_v = \frac{|\{xy : x, y \in N(v)\}|}{\binom{|N(v)|}{2}}.$$

There is no standard definition for the clustering coefficient of a vertex of degree 1.

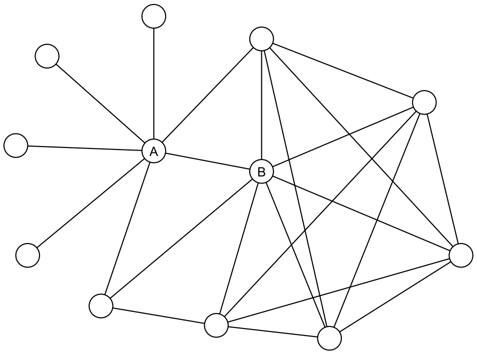


Figure 4.1: Nodes A and B are vertices of highest degree (degree = 7).

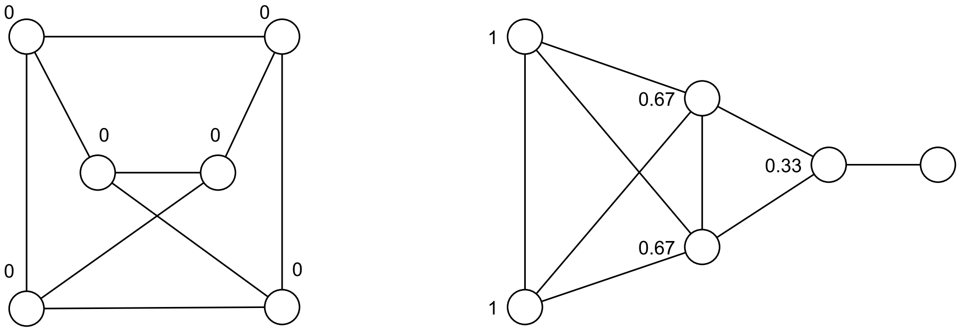


Figure 4.2: Two networks, each with 6 nodes and 9 edges. The network on the left has 0 average clustering coefficient while the network on the right has a high average clustering coefficient.

It has been observed that in real-world networks, nodes tend to cluster themselves more than one would expect from a random collection of edges. That is, the average clustering coefficient in real-world networks is significantly higher than a random graph on the same vertex set and with the same number of edges [140].

Note that the degree and clustering coefficient are strictly *local* properties in the sense that if Figure 4.1 only represents the part of a large network to which A or B are adjacent to and that there are thousands of other nodes attached to the network depicted, the degree-values and clustering coefficients of A and B are unchanged. If Figure 4.1 represents the entire network, then A is, in fact, an important actor since four other vertices rely on A as the only connection into the network. A centrality measure that captures this importance of a node with respect to the entire network topology is the *betweenness centrality* defined by sociologist Linton Freeman in 1977 [53].

Definition 4.2. Betweenness Centrality. The *betweenness centrality* of a node v is $g(v)$ defined as:

$$g(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

where σ_{st} is the number of shortest paths from s to t and $\sigma_{st}(v)$ is the number of shortest paths from s to t that pass through v.

Often, betweenness centrality measures are *normalized* by dividing all betweenness centrality values by the largest betweenness centrality value since a score of, say, 5 means very different things if 5 is the largest value in the network or if it is overshadowed by a score of 20.

In Figure 4.1 where vertices A and B have equal degree centrality, if the depicted graph is the entire network, then the normalized betweenness centralities of the two labeled vertices are $g(A) = 1.0$ and $g(B) = 0.36$, indicating that vertex A is a central communication hub for the vertices around it.

A similar definition of betweenness for edges has been defined and applied to a variety of network analysis applications, such as community-finding [62]. Its definition is analogous to that of betweenness for vertices.

Definition 4.3. Edge Betweenness Centrality. The *edge betweenness centrality* of an edge uv in a connected graph is $g(uv)$ defined as:

$$g(uv) = \sum_{s \neq t} \frac{\sigma_{st}(uv)}{\sigma_{st}}$$

where σ_{st} is the number of shortest paths from s to t and $\sigma_{st}(uv)$ is the number of shortest paths from s to t that pass through the edge uv.

Many other definitions of centrality exist to distinguish types of node or edge importance, for example *page rank* [51] or *closeness centrality* [125].

4.1 Degree Distribution and Power Law

Vertex degrees are usually easy to calculate and visualize. The implication of a node having many connections is meaningful for many aspects of network analysis, and so it may be premature to quickly dismiss the use of degree centrality simply because it is a local property that does not consider the global network topology. Instead, we can gain information about the general structure of a network by looking at the *distribution* of degrees in the network. For a particular network, let $P(k)$ denote the proportion of nodes having degree k.

Definition 4.4. Power Law. A graph has a *power law degree distribution* if

$$P(k) \propto k^{-\gamma} \text{ for some constant } \gamma > 0.$$

(Typically, $2 < \gamma < 3$). A network having a power law degree distribution is sometimes called a *scale-free network*.

A power law degree distribution occurs when there are relatively many high-degree nodes. A large variety of networks evolving from natural processes have been found to exhibit a scale-free structure, notably whenever a *preferential attachment* mechanism for network growth is in play (see Section 4.4).

4.2 The Small-World Phenomenon

In addition to a power law degree distribution, real world networks are often observed to exhibit a *small world* property. That is, two seemingly unrelated individuals in a network are supposedly connected through a path of relatively short length.

As an example of this, the network of movie actors has roughly 1.5 million nodes, where two nodes are joined by an edge if they appear in a movie together. The mathematician Paul Erdős and the famous actor Kevin Bacon have very little in common and are fairly unrelated. But when including documentaries, Paul Erdős was is the biographical film *N is a*

Number (1993) with Tomasz Łuczak; Łuczak was in *The Mill and the Cross* (2011) with Michael York; York was in *Transformers: Revenge of the Fallen* (2009) along with Rainn Wilson, who was also in *Super* (2010) with Kevin Bacon. Hence Paul Erdős is a distance of at most 4 from Kevin Bacon in the network of movie actor collaboration.

The small world phenomenon is often also referred to as *six degrees of separation*, hypothesizing that almost everyone coexisting in a network are separated by a path of at most 6.

As another amusing example, in the network of refereed published papers where individuals are joined by co-authorship, the author of this thesis is joined to physicist Albert Einstein by a path of length 5.

With the understanding that naturally-forming networks tend to have a scale-free topology and thus a significant number of hubs (vertices of large degree), the small world phenomenon is entirely plausible, much in the same way that two small airports s and t are likely connected by a length-3 or length-4 path of direct flights using the hub airports closest to each of s and t for layovers.

Using the definition of graph diameter from Section 1.2.3, we can rephrase the small world phenomenon by simply saying that real-world networks tend to exhibit a diameter very small in comparison to the number of nodes (usually logarithmic). An example of a graph with large diameter would be a *square grid graph* (like a Cartesian grid) on n vertices. Its diameter is $2\sqrt{n} = \Omega(n^{0.5})$, which is polynomial in n. More generally, it has been shown that a random planar graph on n vertices almost surely has a diameter of $n^{\frac{1}{4}+O(1)}$ [21]. The small diameter observed in social networks is witnessed by some random models, such as the large component of an *Erdős-Rényi* graph (see Section 4.4) in which the diameter is bounded by a logarithmic function of the number of vertices [13].

4.3 Graph Diameter

A small graph diameter is a fundamental trait of a close-knit community, as this allows for rapid communication or sharing of resources. Interestingly, terrorist organizations have used this property to conceal their plans by putting together individuals in their networks who are of relatively-far distances away from each other. Krebs writes [90] that even Bin Laden described his strategy in organizing the World Trade Center attacks by using individuals in his network who did not know each other. Furthermore, the groups to work together did not know the other groups. Krebs observes

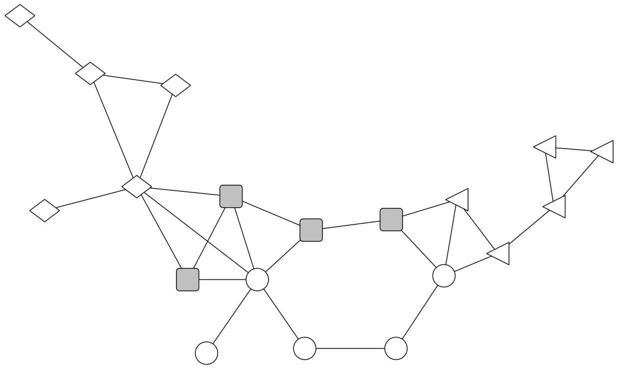


Figure 4.3: A network of 19 individuals, linked by known trusted prior communication, who hijacked and crashed 4 planes in the 9/11 attacks. Diamond nodes crashed into the Pentagon, square nodes crashed in Pennsylvania, circle nodes crashed into WTC South, and triangle nodes crashed into WTC North.

that if a chosen individual is captured or compromised, his social distance away from the others minimizes the damage to the network. We also note that if the terrorist network was known beforehand, observing a potential gathering of those individuals who are mutually-distant may not trigger any warnings.

In order to coordinate their large project, it is known that brief meetings between certain long distance pairs took place, creating temporary shortcuts in the network. Once the coordination was completed, those secret cross-ties were left to go dormant in effort to be unnoticeable.

Krebs writes that 6 additional "shortcut" edges can be added to this network through several known documented meetings, and when these extra edges are added to the network, the diameter of the network changed from 9 to 6. The original mean path length of 4.75 reduced to 2.79, with insignificant changes to the overall edge density (from 16% to 19%) and clustering coefficient (from 0.41 to 0.42). Thus information flow in the network temporarily gained a significant boost while remaining relatively covert.

The selection of these shortcut edges is an interesting problem to look at, namely, which edges can be added to a network in order to reduce the graph diameter the most? A similar, but different, problem would ask how many shortcut edges would have to be added in order to bring the network diameter down to a certain value. The latter problem is exactly Problem 3 in Section 1.2.3. We show here that this computational problem is fixed-parameter hard.

4.3.1 Diameter Augmentation is $W[2]$-hard

For any two vertices x, y in a connected graph, the *distance* between them $dist(x, y)$ is the number of edges in a shortest path joining them. A graph G has *diameter* D if $dist(x, y) \leq D$ for every pair of vertices x, y. The Diameter-D Augmentation problem takes as input a graph $G = (V, E)$ and a value k and asks whether there exists a set E_2 of new edges so that the graph $G_2 = (V, E \cup E_2)$ has diameter D. This problem was first shown NP-hard for $D = 3$ [128] and was later shown to remain hard for the $D = 2$ case [95]. Note that the case of $D = 1$ is trivially polynomial-time solvable as adding an edge between every pair of nonadjacent vertices is necessary.

The proof in [95] reduced a restricted (but still NP-hard [58]) 3-Sat problem to a relaxed dominating set problem (which they called Semi-Dominating Set) which was then reduced to Diameter-2 Augmentation. We provide a reduction to Diameter-2 Augmentation directly from Dominating Set, which not only provides an alternate proof of NP-hardness but also establishes that Diameter-2 Augmentation is $W[2]$-hard.

The Dominating Set problem is:

Problem 7. Dominating Set
Input: A graph $G = (V, E)$ and a positive integer k.
Task: To determine if there exists a set $S \subseteq V$ of size at most k such that for every $v \in V \setminus S$ there is some $s \in S$ where $\{s, v\}$ is an edge.

Reduction from Dominating Set

We reduce Dominating Set to Diameter-2 Augmentation via a *parameterized reduction*. That is, we give a mapping that sends a yes-instance (G, k) of Dominating Set to a yes-instance (G_2, k_2) of Diameter-2 Augmentation where k_2 depends on k alone. Our mapping corresponds to $k_2 = k$.

Problem 8. Diameter-2 Augmentation
Input: A graph $G = (V, E)$ and a positive integer k.
Task: To determine if there exists a set of at most k edges that can be added to G so that the resulting graph has diameter 2.

Let (G, k) be an instance of Dominating Set, where $G = (V, E)$, $|V| = n$. Let $V = \{v_1, v_2, \ldots, v_n\}$. We construct a graph G_2 with two isomorphic copies of G called $G = (V, E)$ and $G' = (V', E')$, and let $V' = \{v_{n+1}, v_{n+2}, \ldots, v_{2n}\}$, such that for every pair i and j:

$$i, j \leq n, \{v_i, v_j\} \in E \iff \{v_{n+i}, v_{n+j}\} \in E'.$$

That is, the mapping $\phi(v_i) = v_{n+i}$ is an isomorphism from G to G'. For $i = 1, \ldots n$, we will refer to the pair v_i and v_{n+i} as *twins* and denote this relationship by $v_i^T = v_{n+i}$ and $v_{n+i}^T = v_i$.

The *open neighbourhood* of a vertex v is the set of vertices adjacent to v, denoted $N(v)$. The *closed neighbourhood* of v is $N[v] = \{v\} \cup N(v)$. In our constructed graph G_2, on each vertex v in V, add the edge $\{v, \phi(u)\}$ for every u in $N[v]$. Also add to G_2 the vertex set Y where there is a vertex $y_{ij} \in Y$ for every pair of indices $1 \leq i, j \leq 2n$, and join y_{ij} to each of v_i and v_j. Add to G_2 an edge between each pair of vertices in Y (so Y induces a clique with $2n$ vertices and $\binom{2n}{2}$ edges.)

Finally, we create in G_2 a vertex z adjacent to every vertex of Y and adjacent to no vertex in $G_1 \cup G_2$, and create a vertex x adjacent to z alone. (See Fig. 1.)

In summary, given a graph $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$, we construct $G_2 = (V_2, E_2)$ such that:

- $V_2 = \{x, z\} \cup Y \cup V \cup V'$, where
- $Y = \{y_{ij} : 1 \leq i, j \leq 2n\}$
- $V' = \{v_{n+1}, \ldots, v_{2n}\}$, and
- $E_2 = E \cup E' \cup \{v_i\phi(u) : 1 \leq i \leq n, u \in N[v]\} \cup \{yy' : \forall y, y' \in Y\} \cup \{v_i y_{ij}, v_j y_{ij} : 1 \leq i, j \leq 2n, i \neq j\} \cup \{zy : \forall y \in Y\} \cup \{xz\}$.

When G has n vertices, G_2 has $2n + \binom{2n}{2} + 2 \in O(n^2)$ vertices, so this reduction is polynomially-sized.

Note that G_2 has diameter at most 3 and also that every pair of vertices in G_2 of distance 3 must be x with some $v_i \in V \cup V'$. It is easy to see that if a dominating set D of G contained k vertices, then the set of edges $\{\{x, d\}, d \in D\}$ forms a diameter-2 augmenting set (also of size k) for G_2. We must prove the converse.

Theorem 4.5. *G has a dominating set of size k if and only if $G_2 = (V_2, E_2)$ has an augmenting set of edges S such that $H = (V_2, E_2 \cup S)$ has diameter 2.*

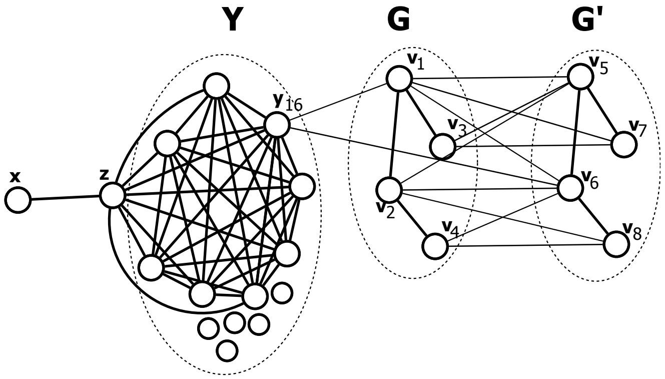


Figure 4.4: A small example of G_2 constructed from $G = P_4$. Only some of set Y is shown.

Before proving this theorem, we will first show that any augmenting edge set A for G_2 with $|A| = k$ that reduces the diameter of G_2 to 2 can be replaced with another augmenting set A', also of size k, such that every edge in A' has the form $\{x, v_i\}$ for some $v_i \in V$.

If an augmenting set of G_2 only contains edges from x to vertices in V we will call it *proper.* We can extract a dominating set of V (and thus of G) from a proper diameter-2 augmenting set A' of G_2 simply by taking all the vertices of V that are adjacent to x in A'.

Say that A is a solution set of edges for Diameter-2 Augmentation on input G_2. We will construct a proper augmenting set of at most the same size as S. After adding the edges in A to G_2, for any vertex $v \in V \cup V'$, there must be a 2-path (or less) joining x to v. If such a 2-path ever passes through the vertex z, we can remove edge $\{z, v\}$ from A and add $\{x, v\}$ to A instead. Note that such an edge-swap can never increase the diameter of the graph. We will provide a sequence of edge-swapping rules to the set A until we arrive at a proper augmenting set A'.

Rule 1. If A has an edge $\{z, v\}$ for any $v \in G_2$ then remove $\{z, v\}$ and add $\{x, v\}$.

Rule 2. If S has any edge $\{x, v_{n+i}\}$ where $v_{n+i} \in V'$, remove $\{x, v_{n+i}\}$ and add the edge $\{x, v_i\}$.

Rule 3. If A has an edge $\{x, y_{ij}\}$ with v_i adjacent to v_j then remove $\{x, y_{ij}\}$ and add the edge $\{x, v_i\}$.

To describe the rest of the rules, we partition $V \cup V'$ into the following dynamic sets. Note that the vertex y_{ij} should be understood to be equivalent to the node y_{ji}.

i) $V_x = \{v : v \in V \cup V', xv \in A\}$

ii) $V^- = \{v_i : v_i \in V \cup V' \wedge v_i \notin V_x \wedge xy_{ij} \in A\}$

iii) $V^+ = \{v : v \in V \cup V', v \notin V_x \cup V^-\}$

Clearly, these three sets are disjoint from each other and their union is exactly $V \cup V'$. To arrive at a proper augmenting set, the edges of A joining vertex x to the set Y will have to be swapped out. The sets V_x, V^-, V^+ are updated after every swap rule is applied. It should be easy to verify that all of the swapping rules will not increase the diameter of the resulting graph $H = (V_2, E_2 \cup A)$.

After applying Rules 1-3, all edges in A are of the form xy for some $y \in Y$ or xv for some $v \in V$. We will use additional rules to remove the edges of the form $xy, y \in Y$.

After applying any of the Rules 4 to 7 (given below), it should be understood that Rule 3, and then Rule 2, may have to be re-applied in order to maintain the invariant that all our edges in A join x to something in $Y \cup V$. Rule 1 will not have to be reapplied after applying it initially. Each rule reduces the number of edges from x to Y, so this process must indeed terminate.

Rule 4. If A has an edge $\{x, y_{ij}\}$ with $v_i \in V_x$ then remove $\{x, y_{ij}\}$ and add the edge $\{x, v_j\}$.

Rule 5. If A has edge $\{x, y_{ij}\}$ and v_i is adjacent to some vertex in V_x then remove $\{x, y_{ij}\}$ and add the edge $\{x, v_j\}$.

Rule 6. If A has two edges $\{x, y_{ab}\}$ and $\{x, y_{cd}\}$ such that v_a is adjacent to v_c then remove $\{x, y_{ab}\}$ and $\{x, y_{cd}\}$ and add $\{x, v_a\}$ and $\{x, y_{bd}\}$.

Proposition 4.6. *If no swap rule can be applied, the set V^- is empty.*

Proof. If there are v_i,v_j in V^- such that $v_iv_j \in E(G_2)$ then Rule 6 could be applied, so we have that V^- is a stable set. If any edge v_iv_j exists in G_2 for $v_i \in V^-$ and $v_j \in V_x$, this would imply Rule 5 can be applied. Thus there are no edges in G_2 joining a vertex in V^- to a vertex in V_x.

Now consider any vertex v in V^-: it must have an adjacent twin vertex (either $\phi(v)$ or $\phi^{-1}(v)$ depending on whether v is in V or V', respectively) and call it v^T.

If v^T is in V_x, then Rule 4 has not been exhausted.

If v^T is in V^- as then either Rule 3 or Rule 7 can be applied, depending on whether the y_{ij} vertex in $N_H(x) \cap N_H(v)$ is or is not $y_{i,n+i}$.

So v^T must be in V^+. Every vertex in V^+ must have a 2-path to x in H, but the vertices in V^+ are not adjacent to any vertex in $N_H(x) \cap Y$, so every vertex in V^+ must be adjacent to a neighbour of x in V_x. Now if v^T is adjacent to some $u \in V_x$ then so is v, which violates the fact that Rule 3 has been exhausted.

Hence no such v can exist, so V^- is empty once these rules can no longer be applied. ■

Once the swap rules can no longer be applied, Proposition 4.6 tells us that all edges in the augmenting set A must be from x to $V_x \subseteq V$ meaning we have arrived at a proper augmenting set.

Now we complete the proof of Theorem 4.5:

Proof. Given any augmenting set for the constructed graph G_2, we apply the swap rules exhaustively until our augmenting set is proper. We can extract a dominating set of size at most k in V_1. In the above notation, this is exactly the set V_x when there are no more edge-swap rules that can be applied. This provides a solution to the dominating set problem on G_1, and so Diameter-2 Augmentation is $W[2]$-hard. ■

4.3.2 Generalization

The following theorem appears in [57], along with an alternate proof of Theorem 4.5.

Problem 9. Diameter-t Augmentation(G, k)
Input: A graph $G = (V, E)$ and a positive integer k.
Task: To determine if there exists a set of at most k edges that can be added to G so that the resulting graph has diameter at most t.

Theorem 4.7. *[57] Diameter-t Augmentation(G, k) is $W[2]$-hard for every integer $t \geq 2$.*

4.3.3 Additional Observations

Consider the following problem, which asks if the diameter of a graph can be improved (i.e. lowered) by adding at most k edges:

Problem 10. Diameter Improvement
Input: A graph $G = (V, E)$ and a positive integer k.
Task: To determine if there exists a set of at most k edges that can be added to G so that the resulting graph has a smaller diameter than G.

The graph G_2 resulting from the reduction from Dominating Set to Diameter-2 Augmentation has diameter 3. Finding an augmenting edge set that improves this graph to diameter 2 will in fact solve the dominating set problem on the original (pre-reduction) graph. This provides a proof that Diameter Improvement is itself $W[2]$-hard (and NP-complete) even when restricted to input graphs of diameter 3.

Theorem 4.8. *The parameterized problem* Diameter Improvement*(G, k) is $W[2]$-hard.*

Another observation about the structure of the constructed graph in the reduction is that it will always contain a dominating clique. Clique-dominated graphs are general enough to include all split graphs. Bertossi [8] showed that Dominating Set is NP-hard for split graphs. Our reduction implies $W[2]$-hardness for diameter augmentation for all clique-dominated input graphs.

Theorem 4.9. *The parameterized problem* Diameter Augmentation*(G, k) is $W[2]$-hard even when G has a dominating clique.*

Recently, independently from our results, the $W[2]$-hardness result of Diameter Augmentation and of Diameter Improvement were proven with a reduction from Set Cover [52].

The next section gives a polynomial time algorithm to optimally solve Diameter Augmentation for a well-studied class of clique-dominated graphs.

4.3.4 Diameter Augmentation for P_4-sparse Graphs

A *P_4-sparse graph* is a graph G for which every set of 5 vertices of G contains at most one P_4 as an induced subgraph. See Section 1.3.4 for more information on these, and for their characterization in terms of thin and thick spider graphs (Lemma 1.2).

If the complement $\overline{G}$ of a graph G is disconnected, then G is necessarily of diameter at most 2.

When a spider has $|K| = |S| = 2$, it is both a thin and thick spider. Consider a thick spider with $|K| = |S| > 2$: any two vertices in S have a common neighbour in K, and so it is easy to verify that the diameter of such thick spiders is 2. Every thin spider has diameter 3.

Solving the diameter augmentation problem for P_4-sparse graphs is thus reduced to the task of finding an augmenting edge set for thin spiders.

Theorem 4.10. *Let $G = (V, E)$ be a thin spider with $|K| = |S| = n$. Then the minimum augmenting edge set A such that $G_2 = (V, E \cup A)$ has diameter 2 is of size $n - 1$.*

Proof. We construct the edge set of size $n - 1$ by choosing $k_1 \in K$ and creating an edge from k_1 to each s_i for $i = 2 \ldots n$. Adding these edges will make k_1 adjacent to every vertex in G, and so the augmented graph has diameter 2.

To see that there can not be an augmenting set of smaller size, we first show that any augmenting set solution on the thin spider can be replaced with another augmenting set solution that contains only edges from S to K. If any augmenting set A contains an edge $\{s_i, s_j\}$, fix i and consider all the edges in A of the form $\{s_i, s_t\}$ for any t. We can replace each $\{s_i, s_t\}$ with $\{k_i, s_t\}$ without increasing the diameter. This process can be continued while there is an edge in A joining two vertices in S.

Next, if the augmenting set joins vertices s_i and s_j by a 2-path through a vertex $r \in R$, then the edges $\{s_i, r\}$ and $\{s_j, r\}$ can be replaced with the two edges $\{s_i, k\}$ and $\{s_j, k\}$ for any $k \in K$. This establishes that any augmenting set A that turns a thin spider into a graph of diameter 2 can be replaced by another augmenting set A' of the same size of A where every edge of A' joins a vertex from S to a vertex in K.

Now consider any augmenting set A' of size $n-2$ where every edge of A' has one endpoint in S and the other endpoint in K. Since there are $n - 2$ edges, there must be two vertices s_p and s_q in S which are not incident with any edge in A'. Their distance must be 3, and so $n-2$ edges are insufficient to improve the diameter. ■

Given a P_4-sparse graph, one can identify the sets S, K, R in linear time simply from the degree sequence [80]. This augmenting set of size $n-1$ can therefore easily be constructed in $O(m + n)$ time.

4.4 Network Models

When testing new ideas on networks it is often desirable to generate a large number of network samples. As mentioned earlier in this chapter, an expected topological property of social networks is the existence of hubs, and so randomly-generated networks should include a heavy-tailed distribution of its vertex degrees.

We note that for applications in which network objects have a location in some space (such as positions on the earth), *random geometric models* [14] have been effective in modeling systems such as wireless networks [83] or disease spreading [9].

4.4.1 The Erdős-Rényi Model

A simple way to randomly generate a network on n nodes is to specify a probability p and for every two nodes in this network, add an edge between them with probability p. Each edge is added independently, so the probability that a particular node v has degree k is

$$P(\deg(v) = k) = \binom{n-1}{k} p^k (1-p)^{n-1-k}.$$

When np is a fixed constant c, this probability approaches $\frac{c^k}{e^c k!}$ as $n \to \infty$, so in this model, the chance that a node has large degree k is quite small.

The clustering coefficient of a vertex in an Erdős-Rényi random graph is easily calculated: recall Definition 4.1 which states that the (local) clustering coefficient of a vertex v is the number of edges in $N(v)$ divided by to total number of possible vertex pairs in $N(v)$. Since in this model, every edge appears independently, the proportion of edges in $N(v)$ that exist has expected value of exactly p. Typically, for large graph generation, p is chosen to be $O(1/n)$, so the amount of local clustering in an Erdős-Rényi random graph is considered too low to model a realistic small-world network.

4.4.2 The Watts-Strogatz Model

In 1998, Watts and Strogatz proposed a model of random graphs to capture the property of being small-world, as well as to exhibit the kind of local clustering observed in real networks. The construction is parameterized by the number of nodes N and the average degree d such that $\ln(N) < d < N$ and a rewiring parameter $\beta \in [0, 1]$. For simplicity, d is assumed to be even.

The model begins with n vertices placed in a cyclic order. Each vertex v is made adjacent to the next $\frac{d}{2}$ vertices in the cyclic order on each side of v. Then, for each vertex u, for each edge ux_i incident with u, rewire it with probability β. To rewire ux_i, choose some vertex x_j uniformly at random such that $x_j \neq u$ and ux_j is not an edge. As β approaches 1, the resulting network looks more like an Erdős-Rényi random graph.

As the initial structure (before rewiring) of the graph model creates many triangles, the clustering coefficient is expected to be high if β is not too close to 1. It has been empirically shown that the small-world behaviour of Watts-Strogatz networks reveals itself for very small values of β (i.e. $\beta = o(1)$), [76] [114].

While this model exhibits the small-world diameter and local clustering expected from real networks, it falls short in capturing the existence of hubs, that is, the heavy-tailed distribution in vertex degrees.

4.4.3 The Barabási-Albert Preferential Attachment Model

Barabási and Albert found, in 1999, that the degree distribution of the World Wide Web (WWW) follows a power law distribution [6], as does a network of movie actors (with two actors being related by being cast in the same movie). They also show a power-law behaviour of vertex degrees for the network built on urban centres where two centres are joined by an edge if there are high-voltage power lines connecting them, and also the network of publications where one publication is (one-way) related to another by way of citation.

Barabási and Albert identify that two of the important properties these networks have are *growth* and *preferential attachment*. That is, if one were to model such networks with a randomly generated graph, the model should allow for a mechanism to add new vertices and for the new edges to be governed by some rule of preference. For example, an Erdős-Rényi random graph in which the number n vertices is specified and the probability p than an edge exists is not sufficient to capture the structural properties of real networks.

This model is sometimes referred to as the *preferential attachment* model or the *BA-model*. Given a network and integer k, we add a new vertex v to the network by attaching v to k existing vertices, randomly selected in such a way that the probability of v being adjacent to an existing u is proportional to the degree of u.

Barabási and Albert showed that this preferential attachment model exhibits a small diameter and a power-law distribution in its vertex degrees. Empirically, the clustering coefficient has been shown to approximate a power law of the network size, which is uncharacteristic of real networks, but this still provides a level of clustering that is above that of Erdős-Rényi graphs.

4.4.4 The Random k-tree Model

While the BA-model has small diameter, a heavy-tailed degree distribution and some clustering, we show in the following that it does not exhibit quite enough clustering when measured in terms of the number of cliques, from triangles to larger ones.

One way to look past vertex degree as a measure of structure is to look at the degree of a pair of vertices. We define a second-order structure called a d-triangle which counts the degree of a pair of vertices which are joined by an edge.

Definition 4.11. The *embeddedness* of an edge $e = uv$ in a graph $G(V, E)$, denoted by $\deg_G(e)$, is defined to be the number of common neighbors of u and v. For an edge $e = uv$ with $\deg_G(e) = d$, the subgraph consisting of the vertices u, v, and their d common neighbors is called a d-triangle.

We also say the **edge embeddedness** of e is d.

We describe one last model here called the k-tree model and compare it to the BA-model in a number of ways.

Starting with an initial k-clique $G^k(k)$, a sequence of graphs $\{G^k(n), n > k\}$ is constructed by adding vertices to the graph, one at a time. To construct $G^k(n+1)$ from $G^k(n)$, add a new vertex v_{n+1} and connect it to the k vertices of a k-clique selected uniformly at random from all the k-cliques in $G^k(n)$. A graph obtained in this manner is called a *random k-tree.*

One of the purposes of this section is to illustrate how the degree sequence of a graph (a "first-order" property) is insufficient to capture many of the internal structures of real social networks and to suggest the mixed k-tree and deleted k-tree models as a way to randomly generate the desirable distributions of first- and higher-order structures.

4.4.5 Cliques and Higher-Order Structures

In our paper [133] the edge-embeddedness and degree distributions were compared for two social networks. One network was created from a sampling of Facebook communication data and a second network was a random sampling of the Orkut friendship network.

For the generated samples, it was found that the degree distribution of Facebook did not have a clear power-law shape while the Orkut network did exhibit such a trend. The edge embeddedness distribution was found to behave similarly to the degree distribution on the respective networks. That is, the edge embeddedness distribution of Facebook did not indicate a power law while the Orkut network did.

This suggests that when designing a random generation process for social networks, it is insufficient to generate networks with a power-law distribution in the degree sequence when the edge embeddedness distribution is ignored.

Indeed, the degree distribution tells very little about graphs in general: the class of graphs which are uniquely defined by degree sequence are the *unigraphs* and these contain threshold ($(P_4, C_4, 2K_2)$-free) graphs. The general graph property of whether a graph is split (partitions into a clique and stable set) is also determinable by degree sequence, but this is a very stringent structure. We argue below that going beyond degree distribution and even beyond edge embeddedness and looking into the distribution of higher-order structures (communities) is necessary to capture the topology of real-world networks.

Since there is no standard definition for a social community, we will follow the definition of Palla et al. [118], as their paper reports on the distribution of community size in real-world networks.

Definition 4.12. A *k-clique community* in a network is defined as a union of all k-cliques that can be reached from each other through a series of adjacent k-cliques, where adjacency means sharing $k-1$ nodes.

One of the intriguing findings in the study of Palla et al. [118] is that the size distribution of the k-clique communities follows a power law in several real-world networks such as the co-authorship networks, word-association networks, and the protein interaction networks. In Figure 4.5, we reproduce plots on the size distribution of k-clique communities obtained [118].

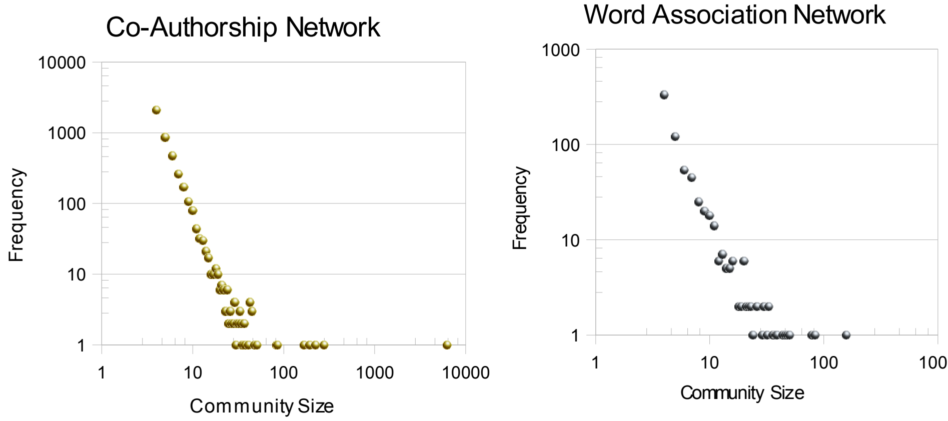


Figure 4.5: Example Networks from Palla et al. with power-law community sizes.

In this section, we show that a simple variant of the random k-tree model is able to capture the characteristic of the community structure much better than other existing models such as the BA model.

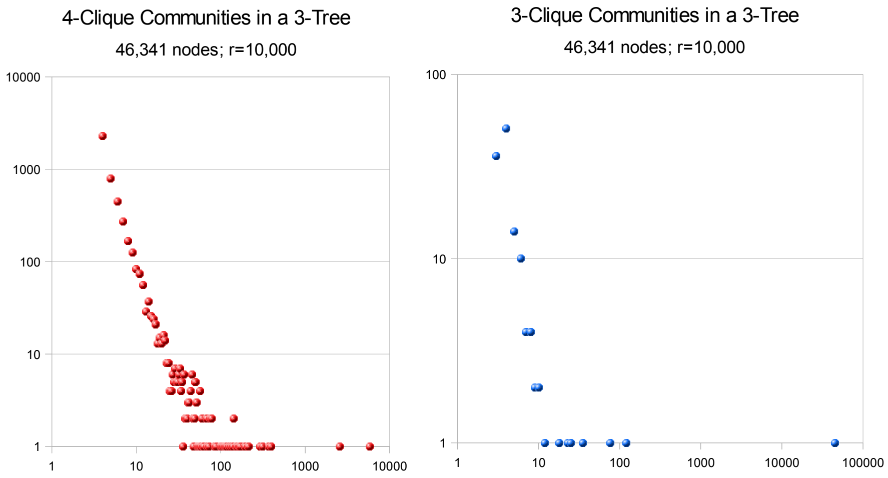


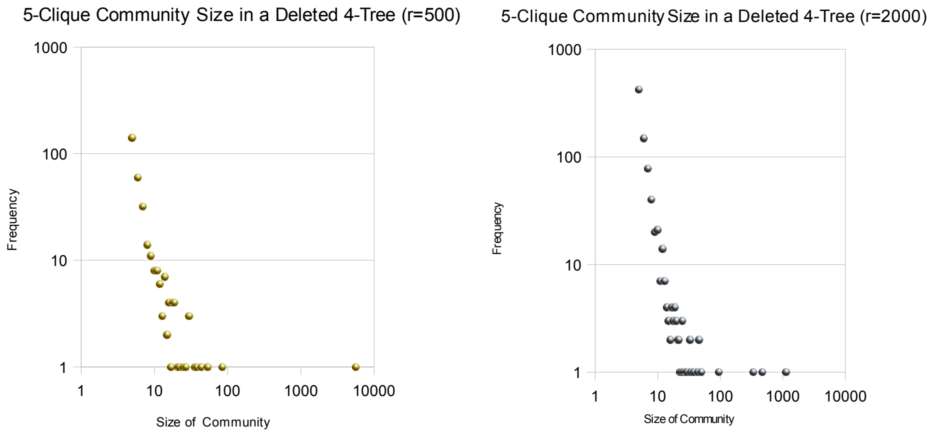
Figure 4.6: K-Clique Communities in a Random Partial 4-Tree.

Partial random k-tree model: A partial k-tree is a subgraph of a k-tree. A random partial k-tree $G^k(n, r)$ is a graph obtained by removing uniformly at random r edges in a random k-tree $G^k(n)$.

When a graph is sufficiently dense, it is expected that all the nodes would exist in one giant K-clique community for relatively small values of K (such as $K = 4$). For relatively large values of K such as K equaling the maximum clique size of a network for instance, the K-clique communities are simply the max cliques. These two extremes create a trivial community structure which reveals nothing about the internal clustering of a network. We are interested in values of K for which there are many distinct K-clique communities, and not just interested in when many vertices are in a K-clique community. It is the distribution of the sizes (in terms of the number of vertices) of the resulting communities that we are interested here, but in order to have a meaningful distribution, there must also be a large number of communities.

The study of Palla et al. [118] found meaningful k-clique communities in real-world networks for $k = 4$.

Using the *clique percolation method* of Palla et al. [118] to find the K-clique community sizes, we analyze the K-clique community sizes of $G^k(n, r)$ for $K \leq k + 1$ with various values for r. When comparing k-trees to BA-


5-Clique Community Size in a Deleted 4-Tree (r=500)

5-Clique Community Size in a Deleted 4-Tree (r=2000)

Figure 4.7: Power Law Community Size in Partial k-trees.

model graphs of similar density, the clique structure in the BA-model is far too sparse to compare the community size distributions. We show how easily partial k-trees can reveal a power law distribution of 4-clique communities while showing how difficult it is to produce any 4-clique distribution in the BA-model.

Aside from directly comparing the partial k-tree model to the BA-model, we look at each model individually as well. For the partial k-tree, we look for the existence of K-clique communities (for $K \leq k$) as r increases. Denote by m the degree of a new vertex added to a network in the BA-model. We set m large in the BA-model to see how far m must be taken in order to find a meaningful K-clique community distribution.

Power-law distributions reveal themselves as straight lines when data is plotted on log-log axes. All the plots in this section will be on log-log scales.

The power law distribution of community size in random partial k-trees is immediately visible even for small values of r and becomes more pronounced as more communities are present. For a k-tree of $2^{15.5} = 46341$ nodes, removing only several hundred edges reveals a power-law distribution in community size and this remains even for r values past 15,000. We show a clear power-law distribution of community size in a 3-tree on $2^{15.5}$ nodes in figure 4.6. The 4-clique communities become apparent with r values near 500, and the figure shows that with a large number of deletions, $r = 10000$, 3-clique communities can simultaneously be found in the same network.

Figure 4.7 shows the distribution of 5-clique community size in a partial

4-tree on 20000 vertices with varying values of r. The plots show $r = 500$ and $r = 2000$ deletions from the same initial k-tree. While both reveal a power law, we also observe a giant community in the $r = 500$ case (the isolated point in the bottom right of the $r = 500$ plot.)

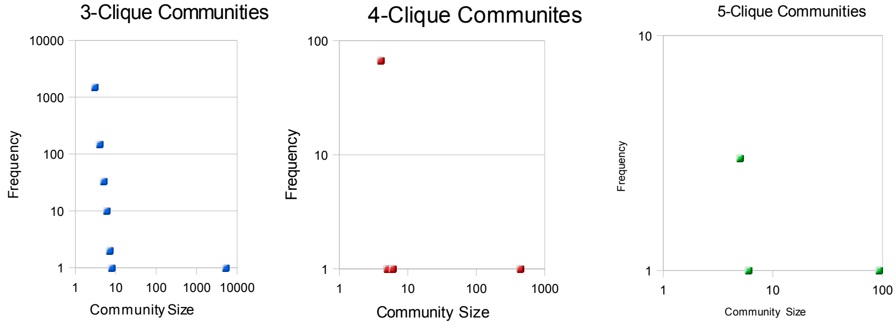


Figure 4.8: K-Clique Communities in BA-model Graphs.

In comparison to the above k-tree models generated with $k = 3$ or $k = 4$, an equally-dense network generated with the BA-model cannot produce similar clustering. Indeed, when a vertex is generated in the BA-model with $m = 4$ is very unlikely that the 4 vertices it attaches to will induce a clique. In order to see any 4-clique community existence, we had to increase m to 9, and even there the structure was very sparse. We produce plots with $m = 10$ in figure 4.8 for networks on 12000 nodes. The BA-model with $m = 10$ could not reveal any significant K-clique community structure for $K > 4$. In order to observe a significant number of 4-clique communities in a BA model network, we had to adjust the m parameter to 13 or higher, but this produces networks of unrelated degree (first-order) measures when compared to k-trees with $k = 4$.

As mentioned above, the BA-model network for m=10 revealed little community structure even for 5-cliques. We generated a BA model network with $m = 20$ and since this creates many maximal cliques, the computation involved for community finding became computationally infeasible for large numbers of nodes. Using a $m = 20$ network on 2000 nodes revealed only a single 10-clique community. The only K-clique community structure found for $K \geq 5$ in this network is shown in figure 4.9.

Following the observations of Palla et al. [118] that real-world networks have a power law distribution of K-clique community size, we give much

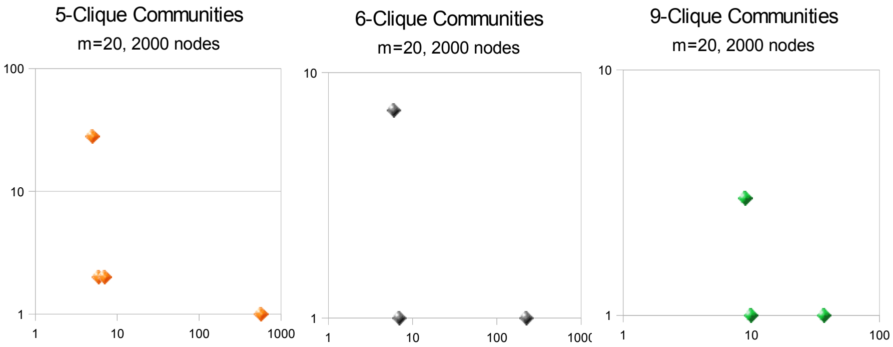


Figure 4.9: K-clique communities in a BA-odel graph with $m = 20$.

evidence here that the partial k-tree model can capture the desired distribution of higher-order structure, and that the strength of the distribution is easily controlled with adjusting the parameter r. We also found that the BA-model does not enjoy the luxury of easily adjusting its structure to properly model a desired community structure.

4.5 A Graph Classes Perspective on Graph Generation

The process of growing a random graph has shown much importance in modeling dynamic networks as many networks allow for growth mechanisms, like people joining a social network or a new paper being added to a library. The BA-model, using preferential attachment, gives rules for how a new vertex can attach to a existing network.

We describe a list of graph operations in common use (in the study of graph classes) for the purpose of growing graphs and reveal implications of certain models, including the k-tree.

i) Add a new vertex adjacent to no existing vertex

ii) Add a new vertex adjacent to only one existing vertex

iii) Add a new vertex adjacent to all existing vertices

iv) Add a new vertex adjacent to a clique

v) Add a new vertex that is adjacent only to all the neighbours of an existing vertex

vi) Add a new vertex that is adjacent only to an existing vertex and all of its neighbours

Starting with a single vertex and repeatedly applying rule (ii) will always generate a tree, and every tree can be generated in that way as well. So rule (ii) completely characterizes a generative model for trees. If we allow any combination of rules (i) and (ii), then this completely characterizes forests (a disjoint union of trees.)

A graph is a threshold graph if the set of vertices can be assigned a real value where, for some fixed threshold value T, two vertices are adjacent in the graph if and only if the sum of their real values exceeds T. Threshold graphs are completely characterized by a generation scheme consisting of rules (i) and (iii).

One of the early graph-generation methods to model the WWW network was proposed in [92], known as a copy model. This model is a random process that repeatedly uses rules (v) and (vi). This rigid copy model was used in a study of biological networks [24], while a more common use of the model is to randomly select a subset of the neighbourhood of an existing vertex chosen uniformly at random. Various forms of the copy model have been shown to have power-law degree distributions [24] [92].

A graph is chordal if every cycle of size 4 or more has a chord. It is known that chordal graphs are completely characterized by the generative scheme defined by rule (iv). We observe that k-trees and mixed k-trees fall under this category: a new vertex added is adjacent only to a clique. When the parameter k (or the bounds of k as in the mixed case) are fixed, these random models do not generate all chordal graphs and so existing results on randomly generated chordal graphs do not necessarily apply. For instance, the paper of Bender et al. [7] shows that under a similar chordal graph generation scheme almost all chordal graphs are split. The authors do this by showing that, as the number of vertices n grows unboundedly, the removal of a largest clique in a random chordal graph almost surely leaves behind isolated vertices (no edges among them.) However, in the k-tree and mixed k-tree model, the maximum clique is easily seen to be bounded by $k+1$ and the removal of this clique removes $\binom{k+1}{2}$ edges, while an n-vertex k-tree has a quantity of edges on the order of nk. As n grows unboundedly, $nk - \binom{k+1}{2}$ is much larger than 0 and so we contrast the result of [7] by observing that while almost all chordal graphs are split, almost no k-tree is split.

As a future consideration, it would be interesting to investigate the random models defined by using various subsets of the vertex-addition rules given above. Rather than just adding a vertex to a graph, many graph classes are characterized by combining two smaller graphs (such as cographs, which are built by disjoint unions and complete joins). Perhaps the notion of adding a collection of nodes at one time can be used in future graph models, perhaps as a way of enforcing certain modularity expectations.

Chapter 5

Bounded Search Tree Methods

A lot of research has been devoted to finding fixed-parameter tractable algorithms for graph modification problems: Guo [70] studied edge deletion to split graphs, chain graphs, threshold graphs and co-trivially perfect graphs; Kaplan et al. [81] studied edge-addition problems to chordal graphs, strongly chordal graphs and proper interval graphs.

Cographs (Section 1.3.3) are an important class of graphs whose study has lead to a general theory of graph decomposition and modularity.

As cographs are exactly the P_4-free graphs, they also provide a generalization to P_3-free graphs (a.k.a. cluster graphs, see Section 1.3.1) which are fundamental to network cluster partitioning methods.

While cographs can be recognized in linear time [31], it is also known that it is NP-complete to decide whether a graph is a cograph with k extra edges [43].

Cai [19] showed fixed-parameter tractability for the edge deletion, edge addition, and edge editing problem to any class of graphs defined by a finite set of forbidden induced subgraphs. The constructive proof implies that k-edge-deletion problems to a class of graphs defined by a finite number of forbidden subgraphs is $O(M^k p(m+n))$ where p is some polynomial and M is the maximum over the number of edges in each of the forbidden induced subgraphs defining the graph class in question. For k-edge-deletions to P_4-free graphs in particular, Cai's result implies an algorithm running in $O(3^k(m+n))$ time. This algorithm would work by finding a P_4: $abcd$ in a graph and branching on the 3 possible ways of removing an edge in order to destroy the P_4 (that is, removing either the edge $\{a,b\}$, $\{b,c\}$ or $\{c,d\}$).

Nikolopoulos and Palios studied the edge-deletion to cograph problem for a graph $G - xy$ where G is a cograph and xy is some edge of G [117]. Lokshtanov et al. study cograph edge-deletion sets to determine whether they are *minimal*, but not a minimum edge-deletion set [98]. To the best of our knowledge, ours is the first study that specifically addresses the edge-deletion problem to cographs. We present a bounded search tree algorithm that solves k-edge-deletion to cographs in $O(2.562^k(m+n))$ time by performing a search until we arrive at a P_4-sparse graph and then optimally solving the remainder of the search space by exploiting the structure of P_4-sparse graphs (see Section 1.3.4).

This chapter presents the first non-trivial algorithm for the cograph edge-deletion problem (running in $O(2.562^k)$) and trivially-perfect edge-deletion problem (running in $O(2.450^k)$ time.) We will give simple algorithms to find minimum vertex-deletion sets to cographs and trivially perfect graphs whose runtime of $O(3.303^k)$ match the existing best methods. We will also illustrate how a careful branching strategy and refined analysis technique can improve the runtime to $O(3.115^k)$ for the cograph vertex deletion problem.

The problems studied in this chapter can be unified in the following way: Given a graph G, we want to delete vertices or delete/add/edit edges in G until the edited graph is in a class $\mathcal{C}$. Consider some larger superclass $\mathcal{L}$ of $\mathcal{C}$. Modifying a graph to the less-restricted class of $\mathcal{L}$ has two benefits: the branching rules on the forbidden subgraphs of $\mathcal{L}$ in order to destroy the forbidden induced subgraphs of $\mathcal{C}$ often yields an improved (smaller) branching factor. Secondly, it may be the case that solving the $\mathcal{C}$-editing problem on a graph of type $\mathcal{L}$ is polytime solvable.

The first step (phase 1) should be performed by making modifications required to transform G into class $\mathcal{C}$, but in such a way that will bring the modified graph into $\mathcal{L}$ first, creating a relaxed stopping condition. If $\mathcal{L}$ is somewhat close to $\mathcal{C}$, it is conceivable that modifying an $\mathcal{L}$-type graph to a $\mathcal{C}$-type graph (phase 2) could be optimally solved in polynomial time, or solved within the same time bound as that required for phase 1.

In this Chapter, we use P_4-sparse graphs for the superclass $\mathcal{L}$ to solve the edge deletion and vertex deletion problems for cographs as $\mathcal{C}$ and quasi-threshold graphs as $\mathcal{C}$.

These results rely on the structure of P_4-sparse graphs, as quasi-threshold and cographs are proper subclasses of P_4-sparse graphs, while the structure of P_4-sparse graphs is simple enough to exploit.

The definition of a spider appears in Section 1.3.4.

Lemma 5.1. *Let G be a spider with body K and feet S. Then every edge $\{k_1, k_2\}$ with $k_1, k_2 \in K$ is in exactly one P_4 in $K \cup S$.*

Proof.
A P_4 cannot contain 3 vertices of K. If G is a thin spider, let each k_i be adjacent to each s_i. The edge $\{k_1, k_2\}$ is only in the P_4 $\{s_1, k_1, k_2, s_2\}$. If G is a thick spider, let each k_i be adjacent to every foot s_j where $i \neq j$. The edge $\{k_1, k_2\}$ is only in the P_4 $\{s_1, k_2, k_1, s_2\}$. □

Algorithm 5: A meta-algorithm paradigm for graph modification problems.

```
Algorithm MetaAlgorithm($G$):
Input: A Graph $G = (V, E)$ and target class $\mathcal{C}$
Output: A minimally-modifed graph $H$ from $G$ such that $H \in \mathcal{C}$

Let $\mathcal{L}$ be an appropriately-chosen superclass of $\mathcal{C}$;
$G' \leftarrow G$;

(Phase 1)
while $G'$ is not in $\mathcal{L}$ do
    Find and edit-out a forbidden substructure which defines class $\mathcal{C}$,
    ideally one that is contained in a forbidden substructure of $\mathcal{L}$;
end

(Phase 2)
Polynomially and optimally solve the rest of the modification
problem of $G'$ of class $\mathcal{L}$ to $\mathcal{C}$;
```

5.1 Edge-Deletion Algorithms

In this section, we give algorithms for two edge-deletion problems.

Problem 11. Cograph Deletion (G, k):
Given graph $G = (V, E)$, does there exist a set S of at most k edges such that $(V, E \setminus S)$ is a cograph?

Problem 12. Trivially Perfect Deletion (G, k):
Given graph $G = (V, E)$, does there exist a set S of at most k edges such that $(V, E \setminus S)$ is a trivially perfect graph?

The idea of the algorithms in this section is to focus on the forbidden subgraphs of P_4-sparse graphs so that efficient branching rules can be designed systematically. The usefulness of this idea depends critically on whether these problems can be solved polynomially on P_4-sparse graphs. We first show how to solve the cograph deletion problem on P_4-sparse graphs in linear time.

5.1.1 Computing Cograph Edge-Deletion Sets on P_4-sparse Graphs in Linear Time

We show that a linear time divide-and-conquer algorithm can be designed to find the minimum cograph deletion set for P_4-sparse graphs.

Definition 5.2. Let G be a graph and $\overline{G}$ be the complement of G. The subgraphs induced by the vertex sets corresponding to the connected components of $\overline{G}$ are called the *co-components* of G. If $\overline{G}$ is connected, then we say that G is *co-connected.*

For a vertex set V_i, write $G[V_i]$ for the induced subgraph from G on vertices V_i.

Proposition 5.3. *Let $G = (V, E)$ be a P_4-sparse graph and $M(V)$ be the size of a minimum edge-deletion set required to turn $G[V]$ into a P_4-free graph.*

i) If G is disconnected with components $V_1, \ldots, V_t$, then

$$M(V) = \sum_{i=1}^{t} M(V_i)$$

ii) If $\overline{G}$ is disconnected with co-components $V_1, \ldots, V_t$, then

$$M(V) = \sum_{i=1}^{t} M(V_i)$$

iii) If G is a spider with head R, body K and feet S, then

$$M(R \cup K \cup S) = M(R) + M(K \cup S).$$

Proof.
(i) This follows from the fact that a P_4 is connected and so any P_4 is in only one connected component, even after some edge deletions.

(ii) It is easy to verify that an edge joining two vertices in separate co-components can not be in a P_4 (or else in the graph complement this would imply a P_4 contains vertices in separate connected components as a P_4 is self-complementary.) After any edge-deletions within a co-component are made, the vertex sets of separate co-components are still completely joined, and so any new P_4s will not include any two vertices in separate co-components.

(iii) Call a *leg* edge any edge joining a vertex $s \in S$ with a vertex $k \in K$, a *head* edge any edge joining some $r_1 \in R$ with some $r_2 \in R$, a *body* edge any edge joining two vertices in K, and call a *neck* edge any edge joining some $r \in R$ with some $k \in K$.

The structural definition of a spider implies that every vertex in K is adjacent to every vertex in $K \cup R$, even after the removal of any leg edges and head edges. Thus a P_4 can never contain an edge $\{r, k\}$ with $r \in R$ and $k \in K$ even after leg and head edge removals. We will show that there is an optimal solution without body edges.

Consider an edge-deletion set E' such that $G - E'$ is a cograph, and let $E'' \subset E'$ be the set of body edges and neck edges in E'. Consider the P_4s in $G - E' + E''$ (the P_4s created when adding E'' back to $G - E'$.) In $G - E' + E''$, K and R are completely joined and K is a clique and so no P_4 uses a neck edge. So any P_4s in $G - E' + E''$ are strictly in $K \cup S$ or strictly in R. Since E' is a cograph deletion set, the induced graph on R in $G - E' + E''$ is P_4-free. In $K \cup S$, the body edges added back may be in a P_4 with two leg edges, and if so, this P_4 will be unique by Lemma 5.1. Adding the body edges from E'' cannot create a P_4 involving a body edge not in E'', so we just concentrate on the unique P_4 that each of these added body edges may have created. By deleting one of these leg edges for each body edge that creates a P_4, we create a new deletion set $E' - E'' + E'''$ where E''' is a set of leg edges and $|E'''| \leq |E''|$, so this new edge deletion set is a solution no larger than E' which does not contain body or neck edges. □

We note that parts (i) and (ii) of Proposition 5.3 apply to any graph G, and not just P_4-sparse graphs.

Lemma 5.4. *Let G be a thin spider with body $K = \{k_1, \ldots, k_{|K|}\}$ and legs $S = \{s_1, \ldots, s_{|K|}\}$, and $\{s_i, k_j\}$ is an edge if and only if $i = j$. Then a minimum cograph edge-deletion set for $K \cup S$ is $\{\{s_i, k_i\}, i = 1 \ldots |K| - 1\}$.*

Proof.
It is obvious for $|K| = 2$, so assume that $|K| > 2$. Since K is a clique and S is stable, every P_4 in $K \cup S$ has its endpoints in S. Furthermore, every pair of vertices in S are in a unique P_4. Deleting any $|S| - 1$ thin legs will clearly destroy all of the P_4s, so this edge-deletion set is indeed a cograph edge-deletion set. To see that it is of minimum size, assume there is a deletion set of size $|K| - 2$ or less in which two legs are not part of the deletion set. Let these two legs be $\{s_1, k_1\}$ and $\{s_2, k_2\}$ and call them "permanent" in this case. Since $\{s_1, k_1, k_2, s_2\}$ is a P_4 and the edges $\{s_1, k_1\}$ and $\{s_2, k_2\}$ are not in the deletion-set, it must be that $\{k_1, k_2\}$ is in the deletion set. There at most $|K| - 3$ other edges in the deletion set. Now $\{s_1, k_1, k_j, k_2\}$

induces a P_4 for every $j = 3 \dots |K|$. This means that the permanent edge $\{s_1, k_1\}$ is still in $|K| - 2$ P_4s and every pair of these P_4s have distinct edges aside from $\{s_1, k_1\}$. Thus it is impossible to destroy all of these remaining P_4s with only $|K| - 3$ additional deletions or less. □

Lemma 5.5. *Let G be a thick spider with body $K = \{k_1, \dots, k_{|K|}\}$ and feet $S = \{s_1, \dots, s_{|K|}\}$, and $\{s_i, k_j\}$ is an edge if and only if $i \neq j$. Then a minimum cograph edge-deletion set for $K \cup S$ is $\{\{k_i, s_j\}, i < j\}$.*

Proof.
Every edge in $K \cup S$ is in exactly one P_4: an edge $\{k_i, k_j\}$ is only in the P_4 $\{s_j, k_i, k_j, s_i\}$ and any edge $\{s_i, k_j\}$ is only in the P_4 $\{s_i, k_j, k_i, s_j\}$ so the number of P_4s in $K \cup S$ is $\binom{|S|}{2}$, and since no two of these P_4s share an edge, at least $\binom{|S|}{2}$ deletions are required. Consider the edge set $T = \{\{k_i, s_j\}, i < j\}$. When deleting T from $K \cup S$, K is still a clique and S is still a stable set, and so if there is any P_4 in $(K \cup S) \setminus T$, its endpoints must still be in S. But after deletion of T, we have that the neighbourhood of s_i is $N(s_i) = \{k_{i+1}, \dots, k_{|K|}\}$ which means that $N(s_i) \subset N(s_j)$ for all $i > j$, and so no two vertices in S can be the endpoints of a P_4. So T indeed destroys all the P_4s in $K \cup S$ and since $|T| = \binom{|S|}{2}$, this is a minimum set. □

Theorem 5.6. *Algorithm 6 correctly solves the cograph edge-deletion problem for P_4-sparse graphs and can be implemented in $O(m + n)$ time.*

Proof.
The correctness of Algorithm 6 follows from Lemma 5.4, Lemma 5.5 and Proposition 5.3.

Algorithm 6 can be implemented in linear time, as the spider structure of P_4-sparse graphs can be identified in linear time [79]. Identifying the connected or co-components can also be done in linear time, as these types of partitions are special cases of the more general notion of a *homogeneous set* or *module*, and there are a number of modular decomposition algorithms running in linear time [105], [33]. □

Our algorithm to find cograph edge-deletion sets in P_4-sparse graphs is presented in Algorithm 6. This is an example of a realization of Phase 2 of the meta-algorithm, Algorithm 5.

5.1.2 A Bounded Search Tree Algorithm for Cograph Edge-Deletion

The bounded search tree algorithm (Algorithm 7) finds 5-vertex subsets that induce at least 2 P_4s, branches on the possible ways of destroying the

Algorithm 6: Cograph edge-deletion algorithm for P_4-sparse graphs.

```
Algorithm Spider($G$):
Input: A $P_4$-Sparse Graph $G = (V, E)$
Output: A set $T \subset E$ such that $(V, E \setminus T)$ is a $P_4$-free graph

if $G$ (or $\overline{G}$) is disconnected then
    Let $C_1, \dots, C_t$ be the components or co-components of $G$;
    $T \leftarrow \bigcup_{i=1}^{t}$Spider($C_i$);
end
$G$ is a spider with $K = \{k_1, \dots, k_{|K|}\}$ and $S = \{s_1, \dots, s_{|K|}\}$;
if $G$ is a thin spider then
    Notation: $k_i$ adjacent to $s_j$ if and only if $i = j$;
    Add edge $\{k_i, s_i\}$ to solution set $T$ for every $i = 1, \dots, |K| - 1$;
end
if $G$ is a thick spider then
    Notation: $k_i$ adjacent to $s_j$ if and only if $i \neq j$;
    Add edge $\{k_i, s_j\}$ to solution set $T$ for every pair $i < j$;
end
Return $T \cup$ Spider($R$);
```

P_4s, and then finally arrives at a P_4-sparse graph and calls Algorithm 6. This algorithm either terminates with a call to the subroutine (in the case that a spider structure is encountered) or detects a cograph structure early, or else its integer parameter k has been reduced to 0 or less in which case the number of allowed edge-deletions has been exhausted without reaching a cograph. This is an example realization of Phase 1 of the meta-algorithm, Algorithm 5.

Refer to Figure 1.2 for the possible subgraphs the general search algorithm may encounter. We refer to specific edges as they are labeled in Figure 1.2 for each subgraph. The pseudocode description of the general search algorithm branches on one of the deletion sets given in the table below.

Let H be one of the forbidden subgraphs from Figure 1.2. The possible edge-deletion sets to destroy the P_4s in H are:

$H =$

Subgraph	Minimal Edge Deletion Sets
C_5	{a,c}, {a,d}, {b,d}, {b,e}, {c,e}
P_5	{a,d}, {b}, {c}
$\overline{P}_5$	{a,b}, {e,c}, {d,e}, {c,d}, {a,d,f}, {a,c,f}, {b,d,f}, {b,e,f}
4-pan	{a,d}, {a,c}, {b,c}, {b,d}, {e}
co-4-pan	{b,c}, {d}, {e}
fork	{a,b}, {c}, {d}
kite	{a,d}, {a,c,f}, {b,d,f}, {b,c}, {e}

It is routine to verify that any edge-deletion set from each of the 7 induced subgraph cases must contain one of the deletion set cases given in the table. Since every P_4 in the graph must be destroyed with an edge deletion, encountering any of these 7 configurations necessitates the need to apply one of the corresponding deletions.

The runtime of the algorithm is dominated by the size of the search tree. The spider structure can be identified in linear time. When k is the parameter measuring the number of edge deletions left to make, the size $T(k)$ of the search tree produced by this process is found from each branch rule separately:

1. C_5: five branches, each reducing the parameter by 2 gives $T(k) = 5T(k-2)$ and so $T(k) \leq 2.237^k$
2. P_5: $T(k) = 2T(k-1) + T(k-2)$ giving $T(k) \leq 2.415^k$
3. $\overline{P}_5$: $T(k) = 4T(k-2) + 4T(k-3)$ giving $T(k) \leq 2.383^k$
4. 4-pan: $T(k) = T(k-1) + 4T(k-2)$ giving $T(k) \leq 2.562^k$
5. co-4-pan: $T(k) = 2T(k-1) + T(k-2)$ giving $T(k) \leq 2.415^k$
6. fork: $T(k) = 2T(k-1) + T(k-2)$ giving $T(k) \leq 2.415^k$
7. kite: $T(k) = T(k-1) + 2T(k-2) + 2T(k-3)$ giving $T(k) \leq 2.270^k$

The size of the search tree is thus upper-bounded by the worst case of deleting P_4s in a 4-pan: $T(k) \leq 2.562^k$.

Theorem 5.7. *Algorithm 7 correctly solves the cograph k-edge-deletion problem in $O(2.562^k(m+n))$ time.*

Proof.
Jamison and Olariu [79] give a linear time recognition algorithm for P_4-sparse graphs. In the case that the graph being tested is not P_4-sparse, the algorithm terminates upon finding a 5-set of vertices isomorphic to one of the forbidden subgraphs shown in Figure 1.2. In $O(m+n)$ time on a general graph, we can find one of the subgraphs in Figure 1.2 or else assert that our graph is P_4-sparse.

□

5.1.3 A Bounded Search Tree Algorithm for Edge-Deletion to Trivially Perfect Graphs

In [70], Guo studied the edge-deletion problem for *complements* of trivially perfect graphs. We know of no prior study of the specific problem of deleting edges to a trivially perfect graph. A naïve solution would find a subgraph isomorphic to either a P_4 or a C_4 and then branch on the possible ways of deleting an edge from that subgraph, resulting in a worst-case search tree of size $O(4^k)$. A minor observation that deleting any one edge from a C_4 always results in the other forbidden subgraph, P_4, allows us to branch on the 6 possible ways of deleting any 2 edges from a C_4. This results in a worst-case search tree of size $O(3^k)$ due to the 3 edges in a P_4.

We use our strategy of branching towards a relaxation class of trivially perfect graphs. The 6 possible ways of deleting 2 edges from C_4 yield a branching factor of $\sqrt{6}^k \leq 2.45^k$, and since removing two edges from any C_4 is necessary to arrive at a (P_4,C_4)-free graph, our algorithm will begin by performing this branching step before running a P_4-sparse recognition algorithm. Then we proceed as in the previous section, finding any P_4-sparse forbidden subgraph and branching on the ways of deleting P_4s and C_4s in it. Once no P_4-sparse obstruction exists, we solve the problem optimally on the resulting specialized structure (a C_4-free P_4-sparse graph.) The branching rules become simpler in that only 5 of the 7 graphs in Figure 1.2 need consideration. In particular, the *4-pan* that caused the bottleneck of Algorithm 7, is no longer considered and this changes the runtime of the process from $O(2.562^k)$ to $O(2.450^k)$.

One main difference in this algorithm from Algorithm 7 is that C_4s are found and destroyed first, and after any of the P_4-sparse deletions are made, the process restarts with looking for C_4s to destroy again. Once the C_4s are destroyed and the resulting graph is P_4-sparse, we proceed with removing edges with edge-deletion algorithm for thin or thick spiders (Algorithm 6).

Algorithm 8: Bounded search tree algorithm finding a trivially perfect edge-deletion set.

```
Algorithm TRIVIALLYPERFECTEDGEDELETION($G, k$)
Input: A Graph $G = (V, E)$ and a positive integer $k$
Output: A set $S$ of edges of $G$ with $|S| \leq k$ where $(V, E \setminus S)$ is
        trivially perfect if it exists, otherwise NO

Initialize $T = \emptyset$;
if $k < 0$ then
    Return NO;
end
if G is trivially perfect then
    Terminate here and return $S$;
end
if There exists $H$ isomorphic to $C_4$ then
    For each of the 6 possible pairs of edges, $T$;
    TRIVIALLYPERFECTEDGEDELETION($G - T, k - 2$);
    $G \leftarrow G + T$;
end
Apply a $P_4$-sparse recognition algorithm;
if G is $P_4$-sparse then
    $T \leftarrow$ SPIDER($G$);
    if $|S| + |T| \leq k$ then
        Terminate here and return $S \cup T$;
    end
    else
        Return NO;
    end
end
else
    A forbidden graph $H$ from Figure 1.2 exists;
    foreach minimal edge-deletion set $E'$ for $H$ do
        $S \leftarrow S \cup E'$;
        $T \leftarrow$ COGRAPHDELETION($G - S, k - |S|$);
        if $T$ == NO then
            $S \leftarrow S \setminus E'$;
        end
    end
    Return NO;
end
```

The correctness of decomposing the edge-deletion problem into separate problems on $K \cup S$ and R depends a proposition similar to Proposition 5.3.

Proposition 5.8. *Let $G = (V, E)$ be a C_4-free graph and $M(V)$ be the size of a minimum edge-deletion set required to turn $G[V]$ into a (P_4, C_4)-free graph.*

i) If G is disconnected with components $C_1, \ldots, C_t$, then

$$M(V) = \sum_{i=1}^{t} M(C_i)$$

ii) If $\overline{G}$ is disconnected, then G is a complete join between a clique and a smaller C_4-free graph, H, and $M(V) = M(V(H))$.

iii) If G is a spider with head R, body K and feet S, then

$$M(R \cup K \cup S) = M(R) + M(K \cup S).$$

Proof.

Case i): If G has more than one connected component, any edge deletions made in one component cannot create a P_4 or a C_4 in a different connected component.

Case ii): Suppose $\overline{G}$ is disconnected. Let H be a set of at least 2 vertices inducing a connected component in $\overline{G}$. Then H induces a C_4-free graph in G since any induced subgraph of a C_4-free graph is C_4-free. Since H is connected in $\overline{G}$, there must be two non-adjacent vertices u, v of H in G. Let x and y be any two vertices not in H. Since H is a connected component in $\overline{G}$, every vertex in H is adjacent to every vertex outside of H. If x and y are not adjacent, then uxvyu is a C_4 in G, which is impossible. So any vertices outside of H must induce a clique in G. It follows, then, that no P_4 in G includes a vertex of $G \setminus H$, and after any edge deletions in H, no P_4 or C_4 can include a vertex of $G \setminus H$. Hence $M(G) = M(H)$.

Case iii): Notice that no C_4 can include a vertex s from S in a spider even after removals of leg edges and head edges since the neighbourhood of s induces a clique. Since K is a clique, and every $k \in K$ is adjacent to every $r \in R$, there can not exist a C_4 in $K \cup R$ unless the C_4 is completely contained in R. So no C_4 contains an edge from R to K. Therefore, any edge $e = \{r, k\}$ with $r \in R$ and $k \in K$ is not in any C_4 in G, and for any subset of leg edges and head edges E' the edge $e = \{r, k\}$ is not in any

C_4 in $G - E'$. Combining this with Proposition 5.3 for P_4s establishes the decomposition.

□

Proposition 5.8 shows us that since all the C_4s are destroyed in the branching stage of TriviallyPerfectEdgeDeletion(G, k), once we arrive at a C_4-free spider, we are free to delete leg edges without creating a new C_4.

The runtime of Algorithm 8 is dominated by the branching rules once again. Encountering a C_4 results in 6 branches which delete 2 edges each. The resulting recurrence is $T(k) = 6T(k-2)$ and so $T(k) \leq 2.450^k$. Having deleted all the C_4s, we no longer include the $\overline{P}_5$ or the 4-pan cases in our analysis. The runtime analysis for the rest remain unchanged: $C_5 : 2.237^k, P_5 : 2.415^k$, co-4-pan: 2.415^k, fork: 2.415^k, kite: 2.270^k. The search tree is thus bounded by the C_4 case of size $O(2.450^k)$. Finding a C_4 directly is a problem that is currently best-achieved using matrix multiplication [89], so this entire process *as described* runs in $O(2.450^k n^\alpha)$ where $O(n^\alpha)$ is the time required for matrix multiplication ($\alpha \leq 2.376$ [29]).

We can, in fact, modify the algorithm to run linearly in n and m by observing that a graph is P_4-free and C_4-free if and only if it is a chordal cograph. By first running a certifying chordal recognition algorithm [135], we can either deduce that there is no C_4 or else find a C_4 or a C_5 or a larger induced cycle (and thus a P_5) and branch on these subgraphs according to the rules we gave, and if the graph is chordal then we apply a P_4-sparse recognition algorithm to find one of the other forbidden induced subgraph, branch on it, and then re-apply the chordal recognition process.

Theorem 5.9. *Finding a trivially perfect k-edge-deletion set can be solved in $O(2.450^k(m+n))$ time.*

5.2 Vertex-Deletion Algorithms

This section shows how our general method can be used to solve vertex-deletion version of our prior two problems:

Problem 13. Cograph Vertex-Deletion (G, k):
Given graph $G = (V, E)$, does there exist a set S of at most k vertices such that $G - S$ is a cograph?

Problem 14. Trivially Perfect Vertex-Deletion (G, k):
Given graph $G = (V, E)$, does there exist a set S of at most k vertices such that $G - S$ is a trivially perfect graph?

5.2.1 Vertex-Deletion to Cographs

Since removing a vertex set S from a graph $G = (V, E)$ is equivalent to taking the induced subgraph on the vertex set $V \setminus S$, these problems are also often named *maximum induced subgraph* problems. In our case of asking if there is a vertex set of size at most k that can be removed to leave behind a cograph, this is equivalent to asking if there is an induced cograph subgraph of size at least $|V| - k$. Removing a vertex from G can never create a new induced subgraph in G, and so deleting vertices to destroy induced subgraphs is commonly modeled as a HITTING SET problem. In this case in which each P_4 maps to a 4-set in a HITTING SET instance, we have the restricted problem of a 4-HITTING SET. Algorithms for such vertex-deletion problems should always be compared against the state-of-the-art algorithms of $d-$HITTING SET if not anything else. d-HITTING SET is a well-studied NP-complete problem which admits fixed-parameter tractable algorithms. The first improved analysis of d-HITTING SET by Niedermeier and Rossmanith [116] give a search tree of size $O(3.30^k)$, and a more detailed and involved analysis by Fernau [49] improves the bound to $O(3.148^k)$. This is the best known bound for 4-HITTING SET to date.

The simple spider structure of P_4 sparse graphs allows us to describe a simple algorithm for the vertex-deletion problem to cographs. The runtime of this simple algorithm matches that of [66] and of [116]. The algorithm in [66] used branching rules that were designed by breaking the P_4s in every subgraph of size t. Testing various values of t deduced that rules based on subgraphs of size 7 yielded the optimal runtime of an algorithm of this sort, with runtime $O(3.30^k)$. Their algorithm builds branching rules from 447 graphs of size 7, while our algorithm only involves seven graphs on 5 vertices (Figure 1.2.)

In the following subsection, we use the analysis technique of [116] to show that the runtime of our bounded search tree algorithm is $O(3.115^k)$, hence improving on Fernau's $O(3.148^k)$. Our runtime could be improved further if we were to use the methods of Fernau [49], but such an analysis is extensive and would sidetrack from the focus of this section.

We describe the subroutine SPIDER VERTEX-DELETION here. The algorithm works in the same way as Algorithm 6, taking as input a P_4-sparse graph and returning the optimal number of vertices to remove in order to break all P_4s in the graph. For thin spiders, every pair of feet is the end-pair of a P_4, and removing any $|S| - 1$ vertices from S will destroy all the P_4s in the body and legs. Removing less than $|S| - 1$ will leave at least two thin legs and hence a P_4, so $|S| - 1$ is necessary.

Since a set of 4 vertices induces a P_4 in a graph G if and only if they induce a P_4 in $\overline{G}$, deleting any $|K|-1$ vertices from K in a thick spider will destroy all the P_4s in $K \cup S$. In either the thin or thick spider case, the subroutine is then applied to head R. This concludes the description of Spider Vertex-Deletion. The correctness of the algorithm is asserted by the following proposition:

Proposition 5.10. *Let $G = (V, E)$ be a spider with head R, body K and feet S. Let $M(V')$ be the minimum number of vertices required to remove from $G[V']$ in order to turn $G[V']$ into a cograph. Then $M(V) = M(R) + M(S \cup K)$.*

Proof.
Deleting vertices from a graph can never create a new P_4. We know from Proposition 5.3 that no P_4 includes vertices from both K and R. Deleting any vertices from $K \cup S$ will not destroy P_4s in R, and vertex deletions from R will not destroy any P_4s in $K \cup S$. Hence $M(V) = M(R) + M(S \cup K)$. □

Corollary 5.11. *The algorithm* Spider Vertex-Deletion *described above correctly solves the cograph vertex deletion problem for spiders in linear time.*

This implies that Spider Vertex-Deletion serves as an implementation of Phase 2 of the meta-algorithm, Algorithm 5. Algorithm 9 serves to fill Phase 1 of the meta-algorithm.

For a general graph, we proceed as in the cograph edge-deletion algorithm. We find P_4-sparse obstructions and branch on the possible ways of deleting vertices to destroy all P_4s, repeating until the remaining graph is P_4-sparse. The pseudocode description is given in 9.

The branching rules for the vertex deletions are given in a table as before:

$H = \left\{ \right.$

Subgraph	Minimal Vertex Deletion Sets
C_5	{1,2}, {1,3}, {1,4}, {1,5}, {2,3}, {2,4}, {2,5}, {3,4}, {3,5}, {4,5}
P_5	{1,5}, {2}, {3}, {4}
$\overline{P}_5$	{1}, {3}, {4}, {2,5}
4-pan	{2}, {4}, {5}, {1,3}
co-4-pan	{3}, {4}, {5}, {1,2}
fork	{3}, {4}, {5}, {1,2}
kite	{2}, {4}, {5}, {1,3}

The runtime of the algorithm is dominated by the branching steps. The runtime $T(k)$ for the C_5 case depends on 10 branches, while each of the other cases have equivalent runtime analysis.

1. C_5: ten branches, each reducing the parameter by 2 gives $T(k) = 10T(k-2)$ and so $T(k) \leq 3.163^k$
2. All others: $T(k) = 3T(k-1) + T(k-2)$ giving $T(k) \leq 3.303^k$

The runtime of this vertex-deletion algorithm is bounded by $O(3.303^k(m+n))$, matching the runtime of the cograph vertex deletion algorithm generated by automated branching rule design [66].

Theorem 5.12. *Algorithm 9 solves the vertex-deletion problem for cographs in $O(3.303^k(m+n))$ time.*

5.2.2 Improvement using Hitting-Set

The 4-Hitting Set algorithm of [116] involves an analysis which counts when a branch choice can be made on a 3-set. Without counting these cases, an algorithm for 4-Hitting Set which only makes choices on 4-sets will have a search tree size of 4^k. By keeping track of when 3-sets are created in the search process and by branching on 3-sets whenever they are available, the authors of [116] are able to improve the upper-bound to the size of the search tree to $O(3.30^k)$.

By using a similar strategy of choosing specific P_4s to branch on, we have shown that Cograph Vertex Deletion can be solved in $O(3.115^k)$ time [109].

Fernau [49] gave an improved analysis to the Hitting Set algorithm, and we believe that a similar analysis would improve our runtime as well. The details, however, are very involved and lengthy, and would sidetrack us from our focus on branching strategies here.

Comment 1. We show here that using a similar technique to that in [116] can improve our search tree size. For an instance (G, k) of cograph vertex-deletion, we will use an implicit instance of 4-Hitting Set where each set of 4 vertices inducing a P_4 in G corresponds to a 4-set. A cograph deletion set for G will correspond to the hitting set of the set of 4-sets.

We adapt the notion of dominance from Hitting Set to that of P_4s: a vertex v *P_4-dominates* u if v exists in every P_4 that u is in.

Following the Hitting Set method, we observe that if v P_4-dominates u, then any hitting set that contains u could be replaced with a hitting set containing v. Using this observation, we *mark* u in the graph to signify that it will not be in our solution. When we encounter P_4s involving u, the P_4 only needs to be considered as a 3-set to hit. Marking u in G is equivalent to removing u in the implicit 4-Hitting Set instance.

Our vertex-deletion algorithm given in the previous subsection applies a P_4-sparse recognition algorithm to find one of the 7 forbidden configurations from Figure 1. We illustrate how to proceed to the branching step when encountering a P_5: $\{v_1, v_2, v_3, v_4, v_5\}$ with v_1 and v_5 as the endpoints:

If we put v_2 in S, remove v_2 from the graph G and reduce the parameter k by 1. Any set in the hitting set instance H containing v_2 is removed. Otherwise (if v_2 is not in S) we mark v_2 in G and remove v_2 from H, possibly creating some 3-sets. If v_3 is put in S, reduce k by 1 and remove v_3 from the graph, as before. Otherwise (if v_3 is also not in S) then mark v_3 in the graph and remove v_3 from H. If v_4 is in S, reduce k by 1 and remove any set containing v_4. Otherwise (if none of v_2, v_3, v_4 are in S) we add v_1 and v_5 to S, remove them from G and reduce the parameter k by 2.

If we first ensure that P_4-dominated vertices have been removed from consideration, some vertices in the P_5 (or analogous forbidden subgraph) may be marked. We do not need to build branches on cases asking if a vertex v is in S if v is already marked. If we encounter a P_4 in one of our forbidden subgraphs consisting of four marked vertices, we can terminate that branch of the search tree and backtrack.

When encountering any of the P_4-sparse forbidden subgraphs: $P_5, \overline{P}_5$, kite, fork, 4-pan, co-4-pan, we have in each case 3 vertices whose removal will break both P_4s in the obstruction, or else two vertices which must be removed together. Call those first 3 vertices the *breaking vertices.* Our process is summarized in the following algorithm:

Algorithm 10: Using hitting-set for cograph vertex-deletion.

```
Algorithm CographVertexDeletionHittingSet($G, k$)
Input: A Graph $G = (V, E)$ and a positive integer $k$

1. If any 3-set has been created, branch on that 3-set. Repeat until there are no more 3-sets;
2. If any vertex is $P_4$-dominated, mark it in $G$ and remove it from the hitting set. Go to step 1.;
3. Find one of $P_5, \overline{P}_5$, kite, fork, 4-pan, co-4-pan.;
4. If there is a $P_4$ all of whose vertices are marked, Stop and backtrack.;
5. Branch on the (up to 3) unmarked breaking vertices using the cases as described above. Go to step 1.;
6. If all three breaking vertices are marked, include the other two vertices in $S$ and go to 1.;
7. If no such subgraph can be found, our graph is an extended $P_4$-sparse graph (See below.) Solve the remainder optimally without search.
```

Let v and v' be breaking vertices encountered after steps 1 and 2 cannot be applied any further. If v is not in any other P_4 besides the two P_4s in the obstruction graph found in step 3, then v is P_4-dominated by v', but this cannot happen if steps 1 and 2 are done to exhaustion. So we have that v must be in another P_4 not involving v'. When branching on v, we consider $v \in S$, in which case v is removed from G, or $v \notin S$ in which case we remove v from H, creating at least one 3-set since we established that v must be in another P_4 not containing v'.

Step 3 can be performed with a linear-time algorithm recognizing ($P_5, \overline{P}_5$, kite, fork, 4-pan, co-4-pan)-free graphs. These are called *extended P_4-sparse graphs* by Giakoumakis and Vanherpe [61]. This ensures we do not encounter a C_5 at this stage of the process. They showed:

Theorem 5.13. *[61] If C is a C_5 in an extended P_4-sparse graph, then C is a prime module.*

Let C be a C_5 in our graph after reaching step 7 of our hitting-set process. Observe that every 4-set of C induces a P_4, so no vertex of C is contained in a nontrivial module or else we will have one of the forbidden subgraphs of extended P_4-sparse graphs which we have already destroyed. Further, C can

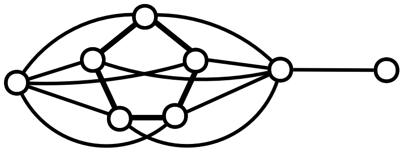


Figure 5.1: An impossible configuration for a C_5 in an extended P_4-sparse graph.

not be a module in some P_4 or else that P_4 extends to one of the forbidden graphs already destroyed (see Figure 5.1.) It must be that C is a set of 5 vertices inducing a 5-cycle and not overlapping with any other P_4. Since C does not intersect with any other existing P_4s left in G, we are free to choose any two vertices of C to add to S and delete from G.

After deleting every C_5 from the extended P_4-sparse graph, we have a conventional P_4-sparse graph and we proceed with vertex deletions for spiders using Spider Vertex-Deletion described in the previous subsection.

Let $T(k)$ be the number of leaves in a search tree of our vertex deletion problem, and let $B(k)$ be the number of leaves in a search tree for this problem whose root branched on a 3-set. Step 4 of Algorithm 10, can (at worst) branch on each of the breaking vertices. If the first vertex is put in S, we reduce k by 1 and so we have a $T(k-1)$ branch. If we assume the first vertex is not in S and select the second vertex to be in S, the parameter decreases by 1. Since this first vertex is not P_4-dominated (or else it would have been marked), it must be in another P_4 and so marking the it will create at least one 3-set in H, giving a $B(k-1)$ branch. Along the same lines, if we choose the third breaking vertex, we arrive at another $B(k-1)$ branch. In the final case of deleting the two non-breaking vertices, we create a $T(k-2)$ branch. Together this puts an upper bound on $T(k)$ of $T(k-1)+2B(k-1)+T(k-2)$.

Similarly, when branching on a 3-set, $B(k) \leq T(k-1) + 2B(k-1)$. Together, these two recurrences give a simultaneous system from which one can show $B(k) \leq 3.115^k$ and $T(k) \leq 1.115B(k)$ with a straightforward induction proof.

Theorem 5.14. *Algorithm 10 solves the cograph vertex-deletion problem in* $O(3.115^k)$ *time.*

The method of analyzing the search tree size created upon the existence of a 3-set shows that our search tree size is smaller than the $O(3.30^k)$ for

4-Hitting Set found by Niedermeier and Rossmanith [116]. Fernau [49] refines this analysis process by keeping track of the *the number* of $(d-1)$-sets in d-Hitting Set, arriving at $O(3.148^k)$ for 4-hitting set. Specifically, Fernau's analysis involves expressions $T^i(k)$ for $i = 0, 1, 2, 3$ where i is the number of 3-sets in an instance of 4-hitting set (in our case, $B(k)$ is $T^1(k)$.) We are confident that a similar refinement in the analysis of our vertex-deletion algorithm would reveal further gains, but our presented algorithm is already shown to have a smaller search space.

5.2.3 Vertex-Deletion for Trivially Perfect Graphs

Given a graph G, our task now is to find the largest induced trivially perfect subgraph in G. Equivalently, given a value k, we want know whether we can delete at most k vertices in order to turn the graph P_4-free and C_4-free.

In the edge-deletion version of this problem from the previous section, we deleted at least 2 edges from all C_4s in the branching process since 2 edges is necessary, and this was algorithmically appealing as it decreased the parameter by 2. The vertex-deletion problem does not share this luxury: there are 4 vertices in a C_4 and only a single vertex removal is required to turn it into a (P_4, C_4)-free graph. This will result in a more complicated procedure to delete all remaining C_4s in the P_4-sparse graph that remains after the search process.

We will proceed directly to finding the P_4-sparse obstructions and branching on them to turn each one into a (P_4, C_4)-free graph. This yields a worst-case runtime of $O(3.303^k)$, as summarized by the following table for each obstruction graph H:

$H =$

Subgraph	Minimal Vertex Deletion Sets
C_5	{1,2}, {1,3}, {1,4}, {1,5}, {2,3}, {2,4}, {2,5}, {3,4}, {3,5}, {4,5}
P_5	{1,5}, {2}, {3}, {4}
$\overline{P}_5$	{1,2}, {1,3}, {1,4}, {1,5}, {2,3}, {2,4}, {2,5}, {3,4}, {3,5}, {4,5}
4-pan	{2}, {4}, {1,3}, {1,5}, {3,5}
co-4-pan	{3}, {4}, {5}, {1,2}
fork	{3}, {4}, {5}, {1,2}
kite	{2}, {4}, {5}, {1,3}

The runtime for each of these cases is summarized below:

1. C_5: five branches, each reducing the parameter by 2 gives $T(k) = 10T(k-2)$ and so $T(k) \leq 3.163^k$
2. P_5: $T(k) = 3T(k-1) + T(k-2)$ giving $T(k) \leq 3.303^k$
3. $\overline{P}_5$: $T(k) = 10T(k-2)$ giving $T(k) \leq 3.163^k$
4. 4-pan: $T(k) = 2T(k-1) + 3T(k-2)$ giving $T(k) \leq 3^k$
5. co-4-pan: $T(k) = 3T(k-1) + T(k-2)$ giving $T(k) \leq 3.303^k$
6. fork: $T(k) = 3T(k-1) + T(k-2)$ giving $T(k) \leq 3.303^k$
7. kite: $T(k) = 3T(k-1) + T(k-2)$ giving $T(k) \leq 3.303^k$

After all the forbidden configurations of P_4-sparse graphs have been destroyed, we are left with a P_4-sparse graph from which we must delete vertices to destroy the remainder of the P_4s and C_4s. While C_4s do not exist in a thin or thick spider, C_4s will exist across co-components. Namely, if A_1 and A_2 are two non-clique co-components, then any two nonadjacent vertices x_1 and y_1 in A_1 and any two nonadjacent vertices x_2 and y_2 in A_2 will induce a 4-cycle. Since each connected component and co-component of a P_4-sparse graph must be a spider, every induced 4-cycle must be the type that crosses non-clique co-components.

In order for this P_4-sparse graph to be C_4 free, all but one of the co-components must be a clique. The only P_4s that will be left to delete will be those strictly in the non-clique co-component. To determine the optimal way at arriving at this point, let us introduce some notation: for a P_4-sparse graph G, let $A_1, A_2, \ldots, A_t$ be the co-components of G. Let $\omega_i = \omega(A_i)$ be the size of a maximum clique in A_i, and η_i be the size of a minimum cograph vertex-deletion set, as found by the algorithm Spider Vertex-Deletion.

We seek to find i such that deleting all co-components $A_j, j \neq i$ into cliques, plus Spider Vertex-Deletion(A_i) is a minimum. That is, we want to find i that minimizes

$$\eta_i + \sum_{j \neq i} |A_j| - \omega_j.$$

For a particular G, $\sum |A_i| = n$ is fixed, as is $\sum \omega_i$. We see that the expression above is minimized when i is chosen such that $|A_i| - \omega_i - \eta_i$ is a maximum.

Our algorithm is as follows:

Algorithm 11: Trivially Perfect Vertex-Deletion Algorithm.

```
Algorithm TriviallyPerfectVertexDeletion($(G, k)$)
Input: A Graph $G = (V, E)$ and a positive integer $k$
Output: Yes if there exists a set $S$ of at most $k$ vertices so that
        $G - S$ is trivially perfect, No otherwise.

while $G$ is not $P_4$-sparse do
    Let $H$ be a $P_4$-sparse obstruction subgraph;
    Branch on the possible ways of deleting the $P_4$s and $C_4$s from H;
    Let $k_1$ be the number of vertex deletions made in this stage;
end
$G$ is $P_4$-sparse. Let $A_1, \ldots, A_t$ be the co-components of $G$;
Fix $i$ to be the lowest index maximizing $|A_i| - \omega_i - \eta_i$;
for $j \neq i$ do
    Fix a maximum clique of $A_j$;
    Delete any vertex of $A_j$ which is not in this maximum clique;
end
Let $k_2$ be the number of vertex deletions made in the for-loop;
Let $k_3$ be the number of deletions performed in Spider
Vertex-Deletion($A_i$);
if $k_1 + k_2 + k_3 \leq k$ then
    return Yes;
end
return No;
```

Since maximum cliques can be found in linear time on P_4-sparse graphs, it is clear that this algorithm runs in polynomial time for any fixed k. The runtime is dominated by the exponential factor from the search tree, which was shown to be $O(3.303^k)$.

Theorem 5.15. *Algorithm 11 is a fixed-parameter tractable algorithm which solves the vertex-deletion problem for trivially perfect graphs in $O(3.303^k)$.*

Of course, a hitting set-style improvement similar to the previous section could be applied here.

5.3 Summary

This chapter focused on solving graph modification problems with the fixed-parameter tractability paradigm of a bounded search tree. Finding graph structures that are clique relaxations was motivated in Chapter 2. Each problem studied in this chapter serves as a way of finding a closest or largest clique-relaxation structure and expressed as graph modification problems.

We developed new and improved algorithms for many problems here, all falling within the structure of the meta-algorithm presented as Algorithm 5.

The main contribution (in the author's opinion) of this chapter is the systematic modularization of solving modification problems to a graph class by using a generalization of the target graph class to both: (1) shorten the depth of search trees and (2) design efficient branching rules that handle multiple forbidden configurations at once, thereby reducing the branching factor of search trees.

The specific results in this chapter will be summarized in Chapter 6.

Algorithm 7: Bounded search tree algorithm computing a cograph edge-deletion set.

```
Algorithm CographDeletion($(G, k)$)
Input: A Graph $G = (V, E)$ and a positive integer $k$
Output: A set $S$ of edges of $G$ with $|S| \leq k$ where $(V, E \setminus S)$ is a
        cograph if it exists, otherwise No

Initialize $T = \emptyset$;
if $k < 0$ then
    Return No;
end
if $G$ is a cograph then
    Terminate here and return $S$;
end
Apply a $P_4$-sparse recognition algorithm;
if $G$ is $P_4$-sparse then
    $T \leftarrow$ Spider($(G)$);
    if $|T| \leq k$ then
        Terminate here and return $S \cup T$;
    end
    else
        Return No;
    end
end
else
    A forbidden graph $H$ from Figure 1.2 exists;
    foreach minimal edge-deletion set $E'$ for $H$ do
        $S \leftarrow S \ \cup E'$;
        $T \leftarrow$ CographDeletion($(G - S, k - |S|)$);
        if $T$ == No then
            $S \leftarrow S \setminus E'$;
        end
    end
    Return No // All branches checked with no solution found;
end
```

Algorithm 9: Bounded search tree algorithm finding a cograph vertex-deletion set.

```
Algorithm CographVertexDeletion($(G, k)$)
Input: A Graph $G = (V, E)$ and a positive integer $k$
Output: A set $S$ of vertices of $G$ with $|S| \leq k$ where $(V \setminus S, E)$ is a
        cograph if it exists, otherwise No

Initialize $T = \emptyset$;
if $k < 0$ then
    Return No;
end
if $G$ is a cograph then
    Terminate here and return $S$;
end
Apply a $P_4$-sparse recognition algorithm;
if $G$ is $P_4$-sparse then
    $T \leftarrow$ SpiderVertexDeletion($(G)$);
    if $|S| + |T| \leq k$ then
        Terminate here and return $S \cup T$;
    end
end
else
    A forbidden graph $H$ from Figure 1.2 exists;
    foreach minimal vertex-deletion set $V'$ for $H$ do
        $S \leftarrow S \cup V'$;
        $T \leftarrow$ CographVertexDeletion($(G - S,\ k - |S|)$);
        if $T$ == No then
            $S \leftarrow S \setminus V'$;
        end
    end
    Return No;
end
```

Chapter 6

Concluding Remarks

This thesis was an attempt to bring in knowledge from the fields of algorithmic graph classes into network analysis, which has been studied by many disciplines in distinctly different ways. Before this work was done, social and complex network analysis made use of trees, cluster graphs, chordal graphs (mostly via treewidth), and only recently have cographs made an appearance in the study of applied networks (see e.g. [151]). We have used quasi-threshold graphs here in the study of social communities and hierarchical organization, and P_4-sparse graphs have made several appearances in the associated algorithmic study of these community-detection problems.

We still feel that the vast knowledge of many hundreds of graph classes [16] (almost 1500 now on http://www.graphclasses.org) have use in defining the future direction of network analysis. For instance, any pair of graph classes comparable by set containment could potentially lead to improved algorithms in the form of Algorithm 5. Additionally, superclasses of cluster graph or quasi-threshold graphs can always be used as a type of clique-relaxation for the purposes of defining community structure, and mostly every class has an interesting structural property that can be used for further analysis of the community structure, just as the comparability-graph equivalence for quasi-threshold graphs serves as a way to find and measure the hierarchical organization of the individuals.

6.1 Summary of Thesis

This thesis began with a survey of definition from graph theory, graph classes, and the algorithmic problems studied on graphs. In Chapter 2, we looked at the problem of finding clique clusters in networks and then we showed the use of quasi-threshold graphs in defining social community as a clique relaxation.

In Chapter 3, we showed how to extend the use of quasi-threshold graphs to the problem of detecting hierarchical organization in directed networks.

In Chapter 4, we briefly surveyed some random network models and argued that although measures such as the degree distributions are important in characterizing an accurate network model, we must go further and understand the distribution of higher-order structures, such as community sizes, to fully capture the the structure of real-world networks.

Chapter 5 builds a general framework for improving search-based algorithms for graph modification problems, and exhibits a number of best-known algorithms for several problems.

6.2 The Key of Contributions of this Thesis

The specific results that are contained in this thesis are summarized below.

- We show that Cluster Deletion can be solved in polynomial time on cographs.
- We show that Cluster Deletion is NP-hard on a class of graphs slightly larger than cographs.
- We define a new structure for social community called *familial groups* and show that they naturally arise from sociologists' previous work.
- We show that quasi-threshold editing is NP-complete, settling an open problem mentioned in 2006 [18], in 2008 [102], and again in 2011 [97].
- We show that the problem of adding edges to a graph in order to make its diameter equal to 2 is $W[2]$-hard. We show that it remains hard even on clique-dominated diameter-3 graphs.
- We implemented and tested familial groups on real-world networks to show that the communities detected were meaningful and/or correct.
- We showed how one might edit directed and weighted networks to find familial group communities and hierarchies.
- We performed an empirical study of random BA-model graphs and showed that the community structure is lacking while random partial k-trees do exhibit a distribution of community sizes consistent with the findings of [118].
- Using P_4-sparse graphs, designed new algorithms with best-known runtimes for vertex deletion and edge deletion modification problems to quasi-threshold graphs and cographs.

6.2.1 Future Considerations

Future Work on Modifying to a Cluster Graph

We showed in Section 2.3 that the Cluster-Deletion problem can be solved in polynomial time when the input graph comes from the class of cographs, while it is NP-hard on a class of graphs slightly larger than cographs. Although there is very little difference between these two classes, it would be interesting to know if one could find a dichotomy theorem that sharpens the statement as to exactly when cluster deletion is polytime solvable with respect to forbidding subgraphs.

We also showed that a greedy maximum clique-finding algorithm would optimally solve cluster deletion on cograph input. This property could apply to superclasses of cographs, even where cluster deletion is NP-hard, since on those classes it might be that detecting maximum cliques would not be a polynomial-time algorithm. Even with the loss of polytime solvability, it would be interesting to determine the class of graphs for which obtaining greedy cliques will solve the cluster deletion problem.

Very recently, Bonomo et al. [15] showed that Cluster Deletion is NP-complete on weighted cographs, in contrast to our result showing it is polytime solvable on unweighted cographs. They further show that Cluster Deletion is NP-complete on chordal graphs, even on P_5-free chordal graphs.

Changing the problem consideration from edge deletions to edge edits makes the problem far more complex. We do not believe cluster editing has been studied on a well-known graph class before, and we do not know the answers to the following problems:

Problem 15. (Open) Given a cograph G and integer k, can we solve Cluster Editing(G, k) in polynomial time?

A weaker version of this problem is:

Problem 16. (Open) Given a quasi-threshold graph G and integer k, can we solve Cluster Editing(G, k) in polynomial time?

Future Work on Familial Groups

Despite the theoretical and computational justifications given for the use of familial groups, there is still work to be done in determining when this method of network clustering is desirable over other existing methods. For instance, [123] found that a scale-free topology of complex networks gives evidence of hierarchically-organized nodes, while networks without such structure (such as those deriving from geographical data) are not hierarchical. It would be interesting to see if the method of familial groups would be consistent with their findings by yielding meaningful results only in scale-free networks.

An aspect of (P_4, C_4) editing is that the edit set is not unique, and we can only speculate at this point how varied the found communities would be in the space of equally-weighted edit solutions. A specific question we can ask is how one could define the intra-communal rank of individuals differently so that the importance of a vertex is perhaps measurable in the original network and not on the found edited graph, which is not unique. Along this line of reasoning, we wonder if there may be an easy way to predict the individuals which will end up as leaders of groups after editing to a quasi-threshold graph.

The computational results shown in Section 2.6 weighed an edge-addition and edge-deletion equally, for each of the C_4 and the P_4. But one of the reasons in justifying the removal of P_4s from a community is that two vertices which are a distance 3 away from each other (that is, beyond the horizon of observability [55]) should likely be in separate close-knit communities. With this interpretation, perhaps it makes sense to consider only edge-deletions in destroying P_4s, or maybe weighing the deletions more favorably than edge additions on those 4 vertices.

Another possible future study is to analyze how much larger the found communities become when relaxing the P_4 restriction to a P_5 restriction, and similarly relaxing the C_4 to a C_5. Many graph classes defined by these forbidden induced subgraphs have already been studied, mostly in terms of their structure, but not necessarily in the context of modifying to such graphs. For example, while every connected (P_4, C_4)-free graph has at least one vertex such that every other vertex in the component is adjacent to it, it is also known that every connected (P_5, C_5)-free graph has a clique in it such that every other vertex is adjacent to some vertex of that clique [34]. As far as we know, a graph modification problem (via edge deletions or edits) has not yet been studied for the class of (P_5, C_5)-free graphs. There are many other possible graph classes that can serve as relaxations to P_4 and C_4-freeness as well.

The computational problem of editing a graph to a nearest (P_4, C_4)-free graph is still rather new. We showed here that it is in fact NP-complete, but this does not rule out fast approximation algorithms or integer linear programming formulations. Even improved exponential-time exact algorithms would be of interest, especially kernelization techniques which could reduce the size of large problem instances.

There have been many studies on inferring global structure from local analysis. With our definition of forbidding certain 4-vertex graphs, this opens the door to new structural analysis possibilities, such as probabilistic modeling techniques used in [27] or [47].

Future Work on Branching Strategies

The algorithms presented in Chapter 5 can all be regarded as instances of Algorithm 5. All of the algorithms of Chapter 5 are deletion algorithms, and edge-edit version of analogous algorithms can also follow the paradigm if (i) all the required branching rules for the minimal edits can be described and (ii) a polytime algorithm solving the editing problem on input from an appropriately-chosen superclass can be found. Indeed, shortly after our result containing algorithms for the cograph vertex and edge deletion and quasi-threshold edge deletion problems, the same approach (also using P_4-sparse graphs) was successfully used to give an improved algorithm for the cograph edge editing problem [96]. Indeed, with the vast literature on structural theorems of graph classes, we would expect that changing the superclass in Algorithm 5 from P_4-sparse graphs to another simple class would lead to many more improved search-based FPT algorithms. Perhaps it is that P_4-sparse graphs are "simple" in the fact that they have bounded cliquewidth that allows one to polynomially-solve modification to its subclasses, in which case it might be worthwhile to consider replacing P_4-sparse graphs to a class of larger, but still bounded, cliquewidth. It would be particularly interesting to see an example of a superclass/subclass pair that can be used in Algorithm 5 where the classes are not defined by a finite induced subgraph characterization, but either by an infinite number of forbidden configurations, or even defined by the admittance of certain vertex orders.

Aside from developing FPT algorithms, this search strategy could also be used to develop faster exponential time algorithms for other problems. For example, finding a maximum clique is $W[1]$-hard and a minimum dominating set is $W[2]$-hard, but these are both solvable in linear time on cographs. One can imagine an implementation of an exponential-time algorithm for these problems which considers adding vertices to its solution set in a manner that destroys P_4s, and once the remaining graph to be searched is P_4-free, the linear-time solver for cographs can be used to efficiently complete that search branch.

Bibliography

[1] T. Vicsek A.-L. Barabási, E. Ravasz. Deterministic scale-free networks. *Physica A*, 299:559–564, 2001. → pages 54

[2] R. D. Alba. A graph-theoretic definition of a sociometric clique. *Journal Mathematical Sociology*, 3:113–126, 1973. → pages 21

[3] B. Balasundaram, S. Butenko, I. V. Hicks, and S. Sachdeva. Clique relaxations in social network analysis: The maximum k-plex problem. *Operations Research*, 59(1):133–142, 2011. → pages 20, 21

[4] B. Balasundaram, S. Butenko, and S. Trukhanov. Novel approaches for analyzing biological networks. *Journal of Combinatorial Optimization*, 10:23–39, 2005. → pages 21

[5] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine Learning*, 56:89–113, 2004. → pages 10, 21, 22

[6] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–5122, 1999. → pages 81

[7] E. A. Bender, L. B. Richmond, and N. C. Wormald. Almost all chordal graphs split. *Journal of the Australian Mathematical Society (Series A)*, 38:214–221, 4 1985. → pages 88

[8] A. A. Bertossi. Dominating sets for split and bipartite graphs. *Information Processing Letters*, 19(1):37–40, 1984. → pages 78

[9] A. N. Bishop and I. Shames. Link operations for slowing the spread of disease in complex networks. *EPL*, 95:18005, 2011. → pages 80

[10] S. Böcker. A golden ratio parameterized algorithm for cluster editing. *J. Discrete Algorithms*, 16:79–89, 2012. → pages 11

[11] S. Böcker, S. Briesemeister, and G. W. Klau. Exact algorithms for cluster editing: Evaluation and experiments. *Algorithmica*, 60(2):316–334, 2011. → pages 20

[12] S. Böcker and P. Damaschke. Even faster parameterized cluster deletion and cluster editing. *Inf. Process. Lett.*, 111(14):717–721, 2011. → pages 11, 23

[13] B. Bollobás. The diameter of random graphs. *Trans. of the American Mathematical Society*, (1):41–52. → pages 71

[14] A. Bonato, J. Janssen, and P. Prałat. A geometric model for online social networks. *Proceedings of the International Workshop on Modeling Social Media*, (4), 2010. → pages 79

[15] F. Bonomo, G. Duran, and M. Valencia-Pabon. Complexity of the cluster deletion problem on chordal graphs, subclasses of chordal graphs, and cographs, 2014. → pages 116

[16] A. Brandstädt, V. B. Le, and J. P. Spinrad. *Graph classes: a survey.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999. → pages 2, 22, 114

[17] H. Broersma, E. Dahlhaus, and T. Kloks. Algorithms for the treewidth and minimum fill-in of HHD-free graphs. In Rolf Möhring, editor, *Graph-Theoretic Concepts in Computer Science*, volume 1335 of *Lecture Notes in Computer Science*, pages 109–117. Springer Berlin / Heidelberg, 1997. → pages 23

[18] P. Burzyn, F. Bonomo, and G. Durán. NP-completeness results for edge modification problems. *Discrete Applied Mathematics*, 154(13):1824–1844, 2006. → pages 35, 52, 115

[19] L. Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Inf. Process. Lett.*, 58(4):171–176, 1996. → pages 9, 22, 39, 90

[20] M.-S. Chang. Algorithms for maximum matching and minimum fill-in on chordal bipartite graphs. In Tetsuo Asano, Yoshihide Igarashi, Hiroshi Nagamochi, Satoru Miyano, and Subhash Suri, editors, *Algorithms and Computation*, volume 1178 of *Lecture Notes in Computer Science*, pages 146–155. Springer Berlin / Heidelberg, 1996. → pages 23

[21] G. Chapuy, E. Fusy, O. Giménez, and M. Noy. On the diameter of random planar graphs. In *Proceedings of the 21st International Meeting on Probabilistic, Combinatorial, and Asymptotic Methods in*

the Analysis of Algorithms, DMTCS Proceedings, volume AM, pages 65–78. → pages 71

[22] J. Chen and J. Meng. A $2k$ kernel for the cluster editing problem. *J. Comput. Syst. Sci.*, 78(1):211–220, 2012. → pages 11

[23] F. P. M. Chu. A simple linear time certifying LBFS-based algorithm for recognizing trivially perfect graphs and their complements. *Inf. Process. Lett.*, 107:7–12, June 2008. → pages 12, 35

[24] F. Chung, L. Lu, T. G. Dewey, and D. J. Galas. Duplication models for biological networks. *Journal of Computational Biology*, 10:677–687, 2003. → pages 88

[25] V. Chvátal and P. L. Hammer. Aggregation of inequalities in integer programming. In B. H. Korte P. L. Hammer, E. L. Johnson and G. L. Nemhauser, editors, *Studies in Integer Programming*, volume 1 of *Annals of Discrete Mathematics*, pages 145 – 162. Elsevier, 1977. → pages 12

[26] V. Chvátal, C. T. Hoàng, N. V. R. Mahadev, and D. de Werra. Four classes of perfectly orderable graphs. *J. Graph Theory*, 11(4):481–495, 1987. → pages 22

[27] A. Clauset, C. Moore, and M. E. J. Newman. Hierarchical structure and the prediction of missing links in networks. *Nature*, 453:98–101, 2008. → pages 47, 118

[28] S. Cook. The complexity of theorem proving procedures. pages 151–158, 1971. → pages 5

[29] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990. → pages 101

[30] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, Mass, 2009. → pages 4

[31] D. G. Corneil, Y. Perl, and L. K. Stewart. A linear recognition algorithm for cographs. *SIAM J. Comput.*, 14:926–934, 1985. → pages 13, 22, 27, 90

[32] B. Courcelle and S. Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000. → pages 13

[33] A. Cournier and M. Habib. A new linear algorithm for modular decomposition. In S. Tison, editor, *CAAP*, volume 787 of *Lecture Notes in Computer Science*, pages 68–84. Springer, 1994. → pages 95

[34] M. B. Cozzens and L. L. Kelleher. Dominating cliques in graphs. *Discrete Mathematics*, 86(1-3):101–116, 1990. → pages 117

[35] P. Damaschke. Fixed-parameter tractable generalizations of cluster editing. In Tiziana Calamoneri, Irene Finocchi, and Giuseppe Italiano, editors, *Algorithms and Complexity*, volume 3998 of *Lecture Notes in Computer Science*, pages 344–355. Springer Berlin / Heidelberg, 2006. → pages 11

[36] P. Damaschke. Bounded-degree techniques accelerate some parameterized graph algorithms. In J. Chen and F. Fomin, editors, *Parameterized and Exact Computation*, volume 5917 of *Lecture Notes in Computer Science*, pages 98–109. Springer Berlin / Heidelberg, 2009. → pages 23

[37] H. A. Dawah, B. A. Hawkins, and M. F. Claridge. Structure of parasitoid communities of grass-feeding chalcid wasps. *Journal of Animal Ecology*, 64:708–720, 1995. → pages 47

[38] A. Dessmark, J. Jansson, A. Lingas, E.-M. Lundell, and M. Persson. On the approximability of maximum and minimum edge clique partition problems. *Int. J. Found. Comput. Sci.*, 18(2):217–226, 2007. → pages 25

[39] G. Dirac. On rigid circuit graphs. *Abhandlungen aus dem Mathematischen Seminar der Universitt Hamburg*, 25:71–76, 1961. → pages 15

[40] S. Donnelly and G. Isaak. Hamiltonian powers in threshold and arborescent comparability graphs. *Discrete Mathematics*, 202(1-3):33 – 44, 1999. → pages 12, 55

[41] R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer-Verlag, 1999. 530 pp. → pages 7

[42] D. A. Mongru Z. N. Oltvai A.-L. Barabási E. Ravasz, A. L. Somera. Hierarchical organization of modularity in metabolic networks. *Science*, 297:1551–1555, 2002. → pages 54

[43] E. S. El-Mallah and C. J. Colbourn. Edge deletion problems: properties defined by weakly connected forbidden subgraphs. *Proc. Eighteenth Southeastern Conference on Combinatorics, Graph Theory, and Computing*, Congressus Numerantium 61:275–285, 1988. → pages 9, 13, 35, 90

[44] L. Euler. Solutio problematis ad geometriam situs pertinentis. *Comment. Academiae Sci. I. Petropolitanae*, 8:128–140, 1736. → pages 1

[45] T. S. Evans. Clique graphs and overlapping communities. *J. Stat. Mech.*, P12037, 2010. → pages 49

[46] M. G. Everett and D. Krackhardt. A second look at krackhardt's graph theoretical dimensions of informal organizations. *Social Networks*, 34(2):159–163, 2012. → pages 54, 55, 57

[47] K. Faust. Triadic configurations in limited choice sociometric networks: Empirical and theoretical results. *Social Networks*, 30(4):273–282, 2008. → pages 118

[48] M. R. Fellows, J. Guo, C. Komusiewicz, R. Niedermeier, and J. Uhlmann. Graph-based data clustering with overlaps. *Discrete Optimization*, 8(1):2–17, 2011. → pages 22

[49] H. Fernau. Parameterized algorithms for d-hitting set: The weighted case. *Theoretical Computer Science*, 411(16):1698–1713, 2010. → pages 102, 104, 108

[50] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75–174, 2010. → pages 20

[51] M. Franceschet. PageRank: Stand on the shoulders of giants. *CoRR*, abs/1002.2858, 2010. → pages 70

[52] F. Frati, S. Gaspers, J. Gudmundsson, and L. Mathieson. Augmenting graphs to minimize the diameter. In *Algorithms and Computation*, pages 383–393. Springer, 2013. → pages 78

[53] L. C. Freeman. A set of measures of centrality based upon betweenness. *Sociometry*, 40:35–41, 1977. → pages 18, 69

[54] L. C. Freeman. Centrality in social networks: Conceptual clarification. *Social Networks*, 1:215–239, 1978. → pages 32

[55] N. E. Friedkin. Horizons of observability and limits of informal control in organizations. *Social Forces*, 62(1):54–77, 1983. → pages 34, 117

[56] Y. Gao, D. R. Hare, and J. Nastos. The cluster deletion problem for cographs. *Discrete Mathematics*, 313(23):2763–2771, 2013. → pages iii, 26

[57] Y. Gao, D. R. Hare, and J. Nastos. The parametric complexity of graph diameter augmentation. *Discrete Applied Mathematics*, 161(10-11):1626–1631, 2013. → pages iii, 8, 77

[58] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979. → pages 73

[59] F. Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Series B*, 16(1):47 – 56, 1974. → pages 16, 22

[60] V. Giakoumakis, F. Roussel, and H. Thuillier. On P4-tidy graphs. *Discrete Math. Theor. Comput. Sci.*, 1(1):17–41, 1997. → pages 22

[61] V. Giakoumakis and J.-M. Vanherpe. On extended p4-reducible and extended p4-sparse graphs. *Theoretical Computer Science*, 180(1-2):269–286, 1997. → pages 106

[62] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002. → pages 18, 42, 49, 69

[63] M. C. Golumbic. Trivially perfect graphs. *Discrete Mathematics*, 24(1):105–107, 1978. → pages 12

[64] M. C. Golumbic. Foreword 2004: The annals edition. In Martin Charles Golumbic, editor, *Algorithmic Graph Theory and Perfect Graphs*, volume 57 of *Annals of Discrete Mathematics*, pages xiii–xiv. Elsevier, 2004. → pages 22

[65] J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Graph-modeled data clustering: Fixed-parameter algorithms for clique generation. In Rossella Petreschi, Giuseppe Persiano, and Riccardo Silvestri, editors, *Algorithms and Complexity*, volume 2653 of *Lecture Notes in Computer Science*, pages 636–636. Springer Berlin / Heidelberg, 2003. → pages 11, 22

[66] J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Automated generation of search tree algorithms for hard graph modification problems. *Algorithmica*, 39(4):321–347, 2004. → pages 102, 104

[67] M. S. Granovetter. The strength of weak ties. *American Journal of Sociology*, 78(6):1360–1380, 1973. → pages 33

[68] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981. → pages 22, 24

[69] S. Guillemot, C. Paul, and A. Perez. On the (non-)existence of polynomial kernels for P_l-free edge modification problems. In Venkatesh Raman and Saket Saurabh, editors, *Parameterized and Exact Computation*, volume 6478 of *Lecture Notes in Computer Science*, pages 147–157. Springer Berlin / Heidelberg, 2010. → pages 13

[70] J. Guo. Problem kernels for NP-complete edge deletion problems: Split and related graphs. In *ISAAC*, pages 915–926, 2007. → pages 35, 90, 98

[71] J. Guo. A more effective linear kernelization for cluster editing. *Theor. Comput. Sci.*, 410(8-10):718–726, 2009. → pages 11

[72] J. Guo, C. Komusiewicz, R. Niedermeier, and J. Uhlmann. A more relaxed model for graph-based data clustering: s-plex cluster editing. *SIAM J. Discrete Math.*, 24(4):1662–1683, 2010. → pages 11

[73] P. L. Hammer and B. Simeone. The splittance of a graph. *Combinatorica*, 1(3):275–284, 1981. → pages 17

[74] S. Hartung, C. Komusiewicz, and A. Nichterlein. On structural parameterizations for the 2-club problem. In *SOFSEM*, pages 233–243, 2013. → pages 21

[75] M. B. Hastings. Community detection as an inference problem. *Phys. Rev. E*, 74:035102, Sep 2006. → pages 20

[76] B. Hayes. Graph theory in practics: Part ii. *American Scientist*, 88(2):104–109, 2000. → pages 81

[77] R. B. Hayward, J. P. Spinrad, and R. Sritharan. Improved algorithms for weakly chordal graphs. *ACM Trans. Algorithms*, 3, May 2007. → pages 22

[78] C. T. Hoàng. Perfect graphs, (Ph.D. thesis). *School of Computer Science, McGill University Montreal*, 1985. → pages 13, 15

[79] B. Jamison and S. Olariu. Recognizing P_4-sparse graphs in linear time. *SIAM J. Comput.*, 21(2):381–406, 1992. → pages 13, 14, 15, 95, 98

[80] B. Jamison and S. Olariu. A tree representation for P_4-sparse graphs. *Discrete Appl. Math.*, 35:115–129, January 1992. → pages 13, 79

[81] H. Kaplan, R. Shamir, and R. E. Tarjan. Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs. *SIAM J. Comput.*, 28(5):1906–1922, 1999. → pages 90

[82] R. M. Karp. Reducibility along combinatorial problems. *Complexity of Computer Computations, Proc. Sympos. IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y.. New York: Plenum*, pages 85–103, 1972. → pages 5, 7, 8, 9

[83] H. Kenniche and V. Ravelomananana. Random geometrix graphs as model of wireless sensor networks. *Computer and Automation Engineering (ICCAE)*, 4:103–107, 2010. → pages 80

[84] T. Kloks, D. Kratsch, and C. K. Wong. Minimum fill-in on circle and circular-arc graphs. *J. Algorithms*, 28(2):272–289, 1998. → pages 23

[85] D. E. Knuth. *The Stanford GraphBase: A Platform for Combinatorial Computing.* Addison-Wesley, Reading, MA, 1993. → pages 46

[86] C. Komusiewicz. *Parameterized Algorithmics for Network Analysis: Clustering & Querying.* PhD thesis, 2011. → pages 10, 24

[87] C. Komusiewicz and J. Uhlmann. Cluster editing with locally bounded modifications. *Discrete Applied Mathematics*, 160(15):2259 – 2270, 2012. → pages 24

[88] D. Krackhardt. Computational organization theory. chapter Graph theoretical dimensions of informal organizations, pages 89–111. 1994. → pages 54, 55, 56, 57, 59

[89] D. Kratsch and J. Spinrad. Between O(nm) and O(n^α). *SIAM J. Comput.*, 36(2):310–325, 2006. → pages 101

[90] V. Krebs. Mapping networks of terrorist cells. *Connections*, 24(3):43–52, 2002. → pages 71

[91] M. Kriványek and J. Morávek. NP-hard problems in hierarchical-tree clustering. *Acta Informatica*, 23(3):311–323, 1986. → pages 10

[92] R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal. Stochastic models for the web graph. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, FOCS '00, Washington, DC, USA, 2000. IEEE Computer Society. → pages 88

[93] C. Lekkerkerker and D. Boland. Representation of finite graphs by sets of intervals on the real line. *Fund. Math.*, 51:45–64, 1962. → pages 16

[94] J. M. Lewis and M. Yannakakis. The node-deletion problem for hereditary properties is np-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980. → pages 13, 16

[95] C. Li, S. T. McCormick, and D. Simchi-Levi. On the minimum-cardinality-bounded-diameter and the bounded-cardinality-minimum-diameter edge addition problems. *Oper. Res. Lett.*, 11:303–308, 1992. → pages 8, 73

[96] Y. Liu, J. Wang, J. Guo, and J. Chen. Cograph editing: Complexity and parameterized algorithms. In B. Fu and D.-Z. Du, editors, *COCOON*, volume 6842 of *Lecture Notes in Computer Science*, pages 110–121. Springer, 2011. → pages 13, 118

[97] Y. Liu, J. Wang, J. Guo, and J. Chen. Complexity and parameterized algorithms for cograph editing. *Theoretical Computer Science*, 461(0):45 – 54, 2012. 17th International Computing and Combinatorics Conference (COCOON 2011). → pages 35, 36, 52, 115

[98] D. Lokshtanov, F. Mancini, and C. Papadopoulos. Characterizing and computing minimal cograph completions. *Discrete Appl. Math.*, 158(7):755–764, 2010. → pages 90

[99] F. Luccio and M. Sami. On the decomposition of networks in minimally interconnected subnetworks. *IEEE Trans. Circuit Th.*, 16:184–188, 1969. → pages 18

[100] R. D. Luce. Connectivity and generalized cliques in sociometric group structure. *Psychometrika*, 15:169–190, 1950. → pages 21

[101] D. Lusseau. The emergent properties of a dolphin social network. *Proceedings of the Royal Society of London Series B-Biological Sciences*, 270:S186–S188, 2003. → pages 47

[102] F. Mancini. Graph modification problems related to graph classes. *Ph.D. thesis, University of Bergen*, 2008. → pages 35, 52, 115

[103] D. Marx. Parameterized Complexity and Approximation Algorithms. *The Computer Journal*, 51(1):60–78, 2008. → pages 7

[104] D. Marx. Chordal deletion is fixed-parameter tractable. *Algorithmica*, 57(4):747–768, 2010. → pages 16

[105] R. M. McConnell and J. Spinrad. Modular decomposition and transitive orientation. *Discrete Mathematics*, 201(1-3):189–241, 1999. → pages 95

[106] R. J. Mokken. Cliques, clubs and clans. *Quality and Quantity*, 13:161–173, 1979. → pages 21

[107] J. Nastos and Y. Gao. A note on the hardness of graph diameter augmentation problems. *CoRR*, abs/0909.3877, 2009. → pages iii

[108] J. Nastos and Y. Gao. A novel branching strategy for parameterized graph modification problems. In *Proceedings of the 4th international conference on Combinatorial optimization and applications - Volume Part II*, COCOA'10, pages 332–346. Springer-Verlag, 2010. → pages iii, 23, 35

[109] J. Nastos and Y. Gao. Bounded search tree algorithms for parameterized cograph deletion: Efficient branching rules by exploiting structures of special graph classes. *Discrete Mathematics, Algorithms and Applications*, 4(1), 2012. → pages iii, 54, 104

[110] J. Nastos and Y. Gao. Familial groups in social networks. *Social Networks*, 35(3):439–450, 2013. → pages iii, 46

[111] A. Natanzon. Complexity and approximation of some graph modification problems. *MSc Thesis, Tel Aviv University*, 1999. → pages 10, 16, 17, 23

[112] A. Natanzon, R. Shamir, and R. Sharan. Complexity classification of some edge modification problems. In *Proceedings of the 25th International Workshop on Graph-Theoretic Concepts in Computer Science*,

WG ’99, pages 65–77, London, UK, 1999. Springer-Verlag. → pages 16

[113] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69, 2004. → pages 47

[114] M. E. J. Newman and D. J. Watts. Renormalization group analysis of the small-work network model. *Physics Letters A*, 263(4-6):341–346, 1999. → pages 81

[115] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford Lecture Series in Mathematics and Its Applications, Oxford University Press, 2006. → pages 7

[116] R. Niedermeier and P. Rossmanith. An efficient fixed-parameter algorithm for 3-hitting set. *J. Discrete Algorithms*, 1(1):89–102, 2003. → pages 102, 104, 108

[117] S. D. Nikolopoulos and L. Palios. Adding an edge in a cograph. In *WG*, pages 214–226, 2005. → pages 90

[118] G. Palla, I. Derényi, I. Farkas, and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435:814, 2005. → pages 83, 84, 86, 115

[119] P. A. Pevzner, H. Tang, and M. S. Waterman. An Eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences of the United States of America*, 98(17):9748–9753, 2001. → pages 1

[120] S. Poljak. A note on stable sets and colourings of graphs. *Commentationes Mathematicae Universitatis Carolinae*, 15(2):307–309, 1974. → pages 25

[121] F. Protti, M. D. da Silva, and J. L. Szwarcfiter. Applying modular decomposition to parameterized cluster editing problems. *Theory of Computing Systems*, 44(1):91–104, 2009. → pages 11

[122] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi. Defining and identifying communities in networks. *Proc. Natl. Acad. Sci. USA*, 101:2658–2663, 2004. → pages 19

[123] E. Ravasz and A.-L. Barabási. Hierarchical organization in complex neworks. *Physical Review E*, 67:026112, 2003. → pages 54, 116

[124] B. A. Reed, K. Smith, and A. Vetta. Finding odd cycle transversals. *Oper. Res. Lett.*, 32(4):299–301, 2004. → pages 17

[125] G. Sabidussi. The centrality index of a graph. *Psychometrika*, 31(4):581–603, 1966. → pages 70

[126] M. Sales-Pardo, R. Guimera, A. A. Moreira, and L. A. N. Amaral. Extracting the hierarchical organization of complex systems. *Proceedings of the National Academy of Sciences*, 104(39):15224–15229, 2007. → pages 54

[127] S. E. Schaeffer. Graph clustering. *Computer Science Review*, 1:27–64, 2007. → pages 20

[128] A. A. Schoone, H. L. Bodlaender, and J. van Leeuwen. Diameter increase caused by edge deletion. *Journal of Graph Theory*, 11(3):409–427, 1987. → pages 8, 73

[129] S. B. Seidman and B. L. Foster. A graph theoretic generalization of the clique concept. *J. of Mathematical Sociology*, 6:139–154, 1978. → pages 21

[130] R. Shamir, R. Sharan, and D. Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1-2):173–182, 2004. → pages 10, 22, 24

[131] P. Smyth and S. White. A spectral clustering approach to finding communities in graphs. *Proceedings of the fifth SIAM international conference on data mining*, 119, 2005. → pages 20

[132] A. Sridharan. Topological features of online social networks. *MSc thesis, Univ. of Victoria*, 2011. → pages iii

[133] A. Sridharan, Y. Gao, K. Wu, and J. Nastos. Statistical behavior of embeddedness and communities of overlapping cliques in online social networks. *CoRR*, abs/1009.1686, 2010. → pages iii, 82

[134] R. E. Tarjan. Decomposition by clique separators. *Discrete Mathematics*, 55(2):221–232, 1985. → pages 16

[135] R. E. Tarjan and M. Yannakakis. Addendum: Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 14(1):254–255, 1985. → pages 101

[136] W. T. Tutte. *Graph Theory*. Cambridge University Press, 2001. → pages 55

[137] F. Wang, H. Du, E. Camacho, K. Xu, W. Lee, Y. Shi, and S. Shan. On positive influence dominating sets in social networks. *Theoretical Computer Science*, 412(3):265–269, 2011. → pages 7

[138] W. Wang, D. Kim, N. Sohaee, C. Ma, and W. Wu. A PTAS for minimum *d*-hop underwater sink placement problem in 2-D underwater sensor networks. *Discrete Mathematics, Algorithms and Applications*, (1):283–289, 2009. → pages 7

[139] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994. → pages 16

[140] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, 1998. → pages 69

[141] M. Weller, C. Komusiewicz, R. Niedermeier, and J. Uhlmann. On making directed graphs transitive. In *Algorithms and Data Structures*, pages 542–553. Springer, 2009. → pages 62

[142] M. Weller, C. Komusiewicz, R. Niedermeier, and J. Uhlmann. On making directed graphs transitive. *Journal of Computer and System Sciences*, 78(2):559–574, 2012. → pages 62

[143] S. H. Whitesides. A method for solving certain graph recognition and optimization problems, with applications to perfect graphs. Technical report, Ithaca, NY, USA, 1982. → pages 16

[144] E. S. Wolk. The comparability graph of a tree. *Proc. Amer. Math. Soc.*, 3:789–795, 1962. → pages 11, 12

[145] E. S. Wolk. A note on the comparability graph of a tree. *Proc. Amer. Math. Soc.*, 16:17–20, 1965. → pages 56

[146] J.-H. Yan, J.-J. Chen, and G. J. Chang. Quasi-threshold graphs. *Discrete Applied Mathematics*, 69(3):247–255, 1996. → pages 33

[147] M. Yannakakis. The node-deletion problem for hereditary properties. *Tech Rep 240., Computer Science Labtratory, Princeton U, Princeton NJ*, 1978. → pages 17

[148] M. Yannakakis. The effect of a connectivity requirement on the complexity of maximum subgraph problems. *J. ACM*, 26(4):618–630, 1979. → pages 9

[149] M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM Journal on Algebraic and Discrete Methods*, 2(1):77–79, 1981. → pages 16

[150] W. W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33:452–473, 1977. → pages 18, 42

[151] E. Zotenko, K. S. Guimarães, R. Jothi, and T. M. Przytycka. Decomposition of overlapping protein complexes: A graph theoretical method for analyzing static and dynamic protein associations. *Algorithms for Molecular Biology*, 1(7), 2006. → pages 114