KERNELIZATION FOR EDGE MODIFICATION PROBLEMS

YUPING KE

PhD

The Hong Kong Polytechnic University

2023

The Hong Kong Polytechnic University

Department of Computing

Kernelization for Edge Modification Problems

Yuping KE

A thesis submitted in partial fulfilment of the requirements for the degree of

of Doctor of Philosophy

March 2023

Abstract

Graph modification problems are significant problems in computer science that have gained considerable attention in recent decades. In this thesis, we focus on edge modification problems, whose task is to make an input graph satisfy some required properties by making small changes to the edges in the graph.

We study kernelization algorithms for edge modification problems toward several graph classes that can be characterized by a finite set of forbidden induced subgraphs and provide small kernels for these problems. A kernelization algorithm is a preprocessing algorithm that reduces the input instances to an equivalent instance with a smaller size in polynomial time. We show that the edge deletion problem toward cluster graphs and the edge addition problem toward paw-free graphs admit linear kernels, a $2k$-vertex kernel for the former one and a $38k$-vertex kernel for the latter one. In addition, we prove that the edge addition problem toward trivially perfect graphs admits a quadratic kernel. We also prove that the edge addition problem and the edge deletion problem toward (pseudo-) split graphs have a kernel with $O(k^{1.5})$ vertices.

Publications arising from the thesis

[1] Gabriel Bathie, Nicolas Bousquet, Yixin Cao, Yuping Ke, and Théo Pierron. “(Sub) linear Kernels for Edge Modification Problems Toward Structured Graph Classes”. In: *Algorithmica* [2022], pp. 1–27.

[2] Yixin Cao and Yuping Ke. “Improved Kernels for Edge Modification Problems”. In: *16th International Symposium on Parameterized and Exact Computation*. 2021.

[3] Yixin Cao, Yuping Ke, and Hanchun Yuan. “Polynomial Kernels for Paw-Free Edge Modification Problems”. In: *16th Annual Conference on Theory and Applications of Models of Computation, TAMC 2020*. Springer Science and Business Media Deutschland GmbH. 2020, pp. 37–49.

[4] Hanchun Yuan, Yuping Ke, and Yixin Cao. “Polynomial kernels for paw-free edge modification problems”. In: *Theoretical Computer Science* 891 [2021], pp. 1–12.

Acknowledgements

I would like to express my sincere gratitude to my advisor, Dr. Yixin Cao. Without his encouragement and support, the thesis could not have been completed, and I would not have been able to emerge from my state of despair and regain my confidence in academic research. His insightful feedback and constructive criticism have been instrumental in shaping my research work and refining my analytical skills. I hope that I did not diappoint him too much.

I would like to express my gratitude to my family, whose unwavering support and encouragement have been a constant source of inspiration and motivation throughout my studies.

It is very fortunate to have a friend like Shenghua Wang. During my PhD studies, we shared an office and developed a deep friendship. I am grateful for his continuous proactive help and support and his constant encouragement. I would also like to thank Yulin Feng for all the help he has given me, for the various useful suggestions he has provided, and for teaching me various technical knowledge. I am grateful to him for enlightening me and making me more confident. Thanks to Guiqiang Mou, Hanchun Yuan, Shenghua Wang, and Yulin Feng for reading and providing valuable feedback on preliminary drafts of this thesis.

I would like to express my deepest gratitude to the participants who took part in my research. Their willingness to share their experiences and perspectives has made my research possible, and I am honored to have had the opportunity to learn from them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation and formulation

Graphs are used to represent a wide range of frameworks across various fields such as physics, biology, linguistics, sociology, and information systems [118, 1, 143, 98]. Usually, these frameworks need to satisfy certain special properties to ensure correctness. However, in real-life situations, these properties may not be satisfied due to data loss or errors in data. In such cases, it becomes necessary to modify the frameworks with minimum cost to satisfy the requirements. This is where graph modification problems arise.

Let's see it more clearly from an example of clustering. It is known that clustering is the task of partitioning instances into some number of groups (clusters) based on their similarity, and it plays a vital role in various areas [87, 163, 90, 146]. Usually, we have two aims for clustering, instances inside a cluster are highly similar to each other, and instances from distinct clusters are different from each other. Then there are various optimization problems about clustering in theory, such as maximizing the similarity and minimizing the difference. Let us see another version of an optimization problem for clustering, which is given in [90]. We can build a graph such that its vertices correspond to instances, and there is an edge between two vertices if the similarity of their corresponding instances is higher than a baseline. Then we want this graph to be composed of vertex-disjoint clusters (clusters in a graph are also called cliques, which means there is an edge between each pair of vertices), we call such a graph a cluster graph, and classically it should be. However, experimental errors can cause the graph to deviate from this ideal. Therefore, we need to add or delete a set of edges with minimum size to the graph to make it a cluster graph. We call this graph modification problem the clustering editing problem. See Figure 1.1 for an example. Ideally, we get a graph such that the vertices with the same color are in a clique, and there is no edge between any two vertices with different colors. However, in practice, we may get a graph like the one in Figure 1.1b.

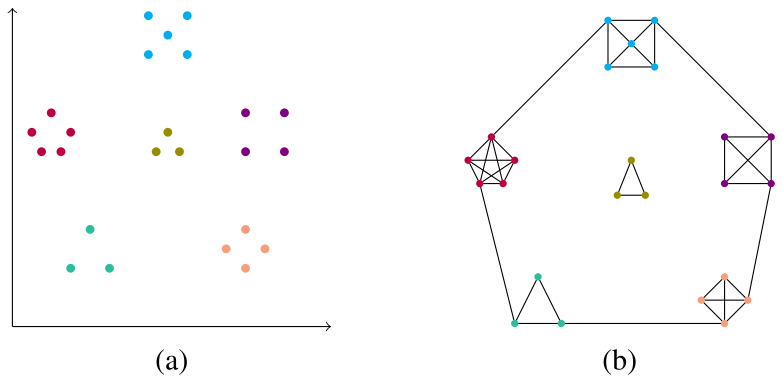


Figure 1.1: An example for clustering. We may get a graph like (b) when we use a graph to represent the framework (since data errors). The edges among vertices with diferent colors should be deleted and the missing edges among vertices with same color should be added.

Numerous problems arising from theory and applications can be formulated as graph modification problems. In theory, one of the most well-studied problems, the vertex cover problem, can be seen as deleting a minimum number of vertices from an input graph to make the resulting graph an edgeless graph. Another famous problem, the feedback vertex set problem can be seen as to deleting a minimum number of vertices from an input graph to make the resulting graph a forest.

Now, we explore several graph modification problems, discussing their applications and motivations. We have an interesting graph modification problem in theory that asks for adding a minimum number of edges to an input graph to transform it into a chordal graph, where every cycle of length greater than three contains a chord [140, 166, 96, 69]. The relationship between the solution to this problem and the solution of a sparse symmetric positive definite system of linear equations by Gaussian elimination (which consists of a sequence of operations performed on the corresponding matrix of coefficients) motivates us to study this graph modification problem [130, 139]. During the process of Gaussian elimination, we may turn zero entries of the matrix into non-zero entries, which would increase the cost of space. Therefore, we want to minimize the number of zero entries turned into non-zero entries. Rose [139] has shown that if we see the matrix of coefficients of the system as the adjacency matrix of a graph, then minimizing the number of zero entries coverted into non-zero entries during the process of Gaussian elimination is equivalent to making the graph to be chordal with the minimum possible number of added edges.

The problem that asks for deleting the fewest vertices from an input graph to make it not contain any cycle with odd length is a classical and extensively researched problem in graph theory [167, 135, 97, 94]. The class of graphs with no odd cycles, also known as bipartite graphs, is a superclass of trees and forests —

two of the simplest nontrivial types of graphs in graph theory. Their significance in graph theory is evident. One notable property of this graph class is that the vertex set of a graph in it can be partitioned into two parts (vertices in every part are nonadjacent to each other). This property makes it a useful tool for modeling real-world scenarios where vertices in one part represents agents, and vertices in the other part represents resources assigned to these agents. Therefore, this graph class is often employed in practice to simulate such situations. Moreover, the graph modification problem related to bipartite graphs has a wide range of applications in various fields, including VLSI design [34] (corresponding to the via minimization problem), computational biology [136] (corresponding to the single-individual haplotyping problem), and register allocation [169] (corresponding to the dual-bank register assignment problem). Thus, there is a strong motivation to study and understand this problem.

In the real world, we often want to increase the network's connectivity so that the failure of any set of a small number of nodes or links in the network does not affect the network's connectivity. Driven by this demand, researchers have started studying the connectivity augmentation problem [60, 157, 22, 71, 120], whose task is to increase the connectivity (vertex/edge connectivity) of an input graph by adding a set of edges with the smallest size. It is a well-known graph modification problem with significant practical applications.

For graph modification problems, we can make modifications to the vertices or edges of a given graph. For the vertex version, Lewis and Yannakakis [110] have shown that the vertex deletion for every nontrivial hereditary graphs is NP-hard. However, for the edge modification problem, we do not have such a dichotomy, although most edge modification problems are known to be NP-complete [166, 44, 124, 144, 20, 116, 63, 164, 46]. In this thesis, we will only consider the edge modification problems.

1.2 A way to solve hard problems

It is known that most graph modification problems are NP-complete, which means that it is unlikely to have a polynomial-time algorithm for solving them, unless $P = NP$ (however, this hypothesis is widely believed to be false). However, in practical scenarios, we usually only need to make a minor modification to a structure to make it satisfy our requirement. As a result, the solution size for graph modification problems tends to be relatively small, even for large input instances. For example, consider the cluster editing problem illustrated in Figure 1.1b, which involves a graph with 24 vertices and 41 edges. By adding two edges among the cyan vertices and deleting five edges, we can obtain a solution of size 7, which is small compared to the size of the input instance.

Therefore, it is natural to use a parameter k to denote the size of a solution and ask whether we can modify at most k edges or vertices to an input graph to obtain the desired graph property. We can then attempt to design an algorithm to solve the problem with time that exponentially depends only on the parameter k. If a problem has such an algorithm, we call this problem a fixed-parameter tractable (FPT) problem, and the algorithm is called a parameterized algorithm. In this way, if we have such a parameterized algorithm, it seems that we can solve those hard problems efficiently. For example, consider the vertex cover problem, which is a classical NP-complete problem. Many researchers [21, 3, 52, 127, 148] have shown an algorithm for this problem that runs in time $O(c^k \cdot n)$, where $c \leq 2$. When

k is small, those algorithms are significantly faster than the brute-force algorithm that runs in time $O(2^n)$. Hence, the existence of such a parameterized algorithm for a hard problem became our first concern.

In numerous works [48, 126, 80, 51, 40], lots of parameterized algorithms are given for various problems. However, many problems have been proved that they are unlikely to admit parameterized algorithms when the parameter is the size of the solutions. In this case, we could change the parameter (not to denote the size of a solution but related to some property of the input graph, e.g., maximum degree) or add more parameters to the problem and try to design an algorithm to solve it with time depending on those parameters [63, 119, 91, 104]. If a problem is proved (probably) not to admit such a parameterized algorithm, then we should spend less effort to attack it (since the hope of progress we could get is tiny). If a parameterized algorithm exists for a problem, then the remaining task is to design an algorithm as efficient as possible. All graph modification problems studied in this thesis have been shown to admit parameterized algorithms, which means they are fixed-parameter tractable.

To design an efficient algorithm for a specific graph modification problem, it is essential to understand the characterizations of the graph class first. One fundamental way to characterize a graph class is through its set of obstructions. The characterizations of numerous graph classes by obstructions have been shown to us, such as perfect graphs, chordal graphs, interval graphs, cluster graphs, and split graphs. We could partition those graph classes into two parts: one contains graph classes with finite obstructions, and the other contains infinite obstructions. Graph classes can be divided into two categories based on their obstructions: those with finite obstructions and those with infinite obstructions. Cai [23] has shown that for graph classes that can be characterized by a finite set of obstructions, the edge modification problems toward them are fixed-parameter tractable. There are many interesting graph classes that can be characterized by a finite number of obstructions, such as cluster graphs, cographs, and trivially perfect graphs are the ones that have been studied widely. However, several important graph classes are characterized by infinite obstructions, such as bipartite graphs, chordal graphs, and (proper) interval graphs. Therefore, Cai's approach cannot be directly applied to develop a fixed-parameter tractable algorithm for edge modification problems toward these graph classes. Fortunately, efficient parameterized algorithms for edge modification problems towards many of these graph classes have been given by multitudinous excellent researchers. For more information, we recommend referring to the survey [39].

1.3 Preprocessing

One way to improve the efficiency of the parameterized algorithms for a problem that is fixed-parameter tractable is by using some techniques, such as dynamic programming [7, 5], tree decomposition [88, 138] and linear programming [142, 147]. Another approach to improve the efficiency of the parameterized algorithms is to do some preprocessing to the input instance to reduce its size. Here preprocessing means dealing with the simple parts of an input instance first and then reducing it to an equivalent instance with a smaller size efficiently (polynomial in time with respect to the input size). We also refer to the preprocessing as kernelization and the minor instance we obtain after doing preprocessing exhaustively as a kernel. The idea of kernelization is intuitive since when faced with a challenging problem,

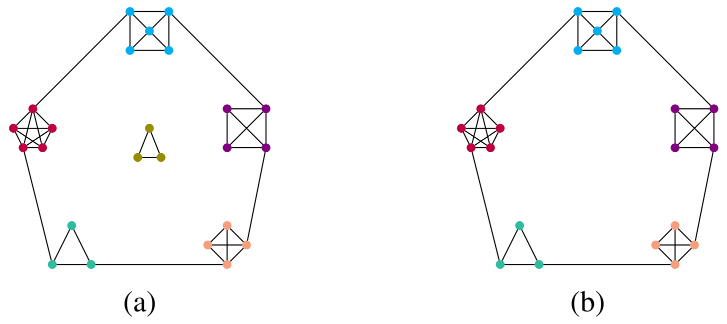


Figure 1.2: Do some preprocessing for the cluster editing problem.

one naturally starts by addressing the simpler aspects. For example, when playing Sudoku, we will first consider the cell such that there are enough known numbers in the row and column to make sure which number should be filled in the cell. In computational theory, for the vertex cover problem (given a graph, ask for finding a set of vertices with minimum size such that every edge in the graph has an endpoint in the vertex set), we first remove vertices with no edges incident to them as they are not part of any optimal solution. See Figure 1.2a for another example, it is easy to see that the three olive vertices should be in a cluster, which means that any optimal solution would not touch any one of them, then it is safe to delete them from the input graph. In the preprocessing subroutine, we could delete the three olive vertices (see Figure 1.2b). Then we obtain a smaller-sized instance.

There are two things we are most concerned about when we do some preprocessing for a problem. One of them is that we need to ensure the correctness of every reduction step, which means that we need to prove that the smaller-sized instance after a reduction step is equivalent to the original instance, where two instances are equivalent means that one instance is a yes-instance if and only if the other instance is a yes-instance. The other one is to analyze the size of the smaller-sized instance after all reduction steps (we want it to be as small as possible). Usually, we aim to get an instance with a polynomial size of the parameter k (polynomial kernel), which means a tiny instance when k is small (it is small in practice).

We already know that for many graph classes that can be characterized by a finite forbidden induced subgraphs, the graph modification problems towards them admit polynomial kernels [79, 84, 85, 33, 28, 10, 82, 54, 75, 43, 56], such as cluster graphs, chain graphs, trivially perfect graphs, threshold graphs, (pseudo-)split graphs, and cographs. We say that a graph is $\mathcal{H}$-free if it does not contain any graphs in $\mathcal{H}$ as an induced subgraph. Then a question comes naturally: whether all $\mathcal{H}$-free graph modification problems admit polynomial kernels, where $\mathcal{H}$ is a finite set of graphs [23, 15, 62].Kratsch and Wahlström [107] answered this question negatively by showing that for a specific graph H on seven vertices (see Figure 1.3), both the edge deletion problem and the editing problem toward H-free graphs do not admit polynomial kernels unless NP $\subseteq$ coNP/poly. Moreover, Kratsch and Wahlström asked whether the edge modification problems always have polynomial kernels for graph classes whose forbidden induced subgraphs consist only of paths, cycles, and cliques. Guillemot et al. [82] answered their question by showing that the edge deletion problem toward P_ℓ-free graphs has no polynomial kernel unless NP $\subseteq$ coNP/poly when $\ell \geq 7$, and the edge deletion problem toward C_ℓ-free graphs has no polynomial kernel unless NP $\subseteq$ coNP/poly when $\ell \geq 4$. They also provided an $O(k^3)$-vertex kernel for the editing problem toward P_4-free graphs (also known as cographs).

Then another question arises: which $\mathcal{H}$-free graphs have polynomial kernels for the edge modification problems? While this question remains open, Cai and Cai [24] have made an outstanding contribution to it. They did comprehensive re-

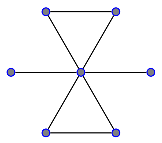


Figure 1.3: A graph H on seven vertices in [107].

search for the nonexistence of polynomial kernels of the edge modification problems for graph classes characterized by forbidding one single graph H, where H is a 3-connected graph, a path, or a cycle. They showed that for a 3-connected graph H, the H-free edge deletion problem and the H-free editing problem have no polynomial kernel if and only if H is not a complete graph, and the H-free completion problem admits no polynomial kernel if and only if H misses at least two edges. When H is a path or a cycle, they showed that the H-free edge deletion problem, the H-free completion problem, and the H-free edge editing problem admit no polynomial kernels if and only if H has at least four edges. For graph classes characterized by a finite set $\mathcal{H}$ of forbidden induced subgraphs, Cai and Cai also proved that the $\mathcal{H}$-free edge deletion problem admits no polynomial kernels if all graphs in $\mathcal{H}$ are 3-connected and there is a graph $H \in \mathcal{H}$ with fewest edges such that one can obtain a graph not in $\mathcal{H}$ by adding an edge to H. More recently, Marx and Sandeep provided additional insights into the existence of polynomial kernels of H-free edge modification problems [117]. They proved that there is a set $\mathcal{F}$ of nine graphs, each with five vertices, such that if the F-free edge editing problem does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly for every $F \in \mathcal{F}$, then for a graph H with at least five vertices the H-free edge editing problem admits a polynomial kernel if and only if H is either empty or complete. They also show that there exists a set $\mathcal{F}$ of nineteen graphs, each with either five or six vertices, such that if the F-free edge deletion problem does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly for every $F \in \mathcal{F}$, then for a graph H with at least five vertices, the H-free edge deletion problem does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly if and only if H is a graph with at least two edges but not complete. For the H-free completion problem, they provided a similar result for it as for the H-free deletion problem.

Based on the results presented in [24, 117], there are very few graphs H such that the edge modification problems toward H-free graphs admit polynomial kernels. However, we still need to work on a general answer to the question of characterizing the graph classes such that the edge modification problems have polynomial kernels.

Naturally, we are also concerned about the efficiency of a preprocessing subroutine (the total time it is used). It is not necessary to do the preprocessing if it is very time-consuming. Therefore, we define a preprocessing algorithm as useful only when it works in polynomial time and returns a minor instance that is equivalent to the given instance [40].

1.4 Our contributions

In this thesis, we study several graph classes that are characterized by a set of finite forbidden induced subgraphs and show polynomial kernels for edge modification problems towards these classes. Let $\mathcal{H}$ denote a set of graphs. We study $\mathcal{H}$-free graphs, where $\mathcal{H}$ is a set of graphs with a size smaller than five (see Figure 1.4). We summarize our contributions in Table 1.1, where the kernels are measured by the number of vertices.

When $\mathcal{H}$ contains one single graph, we use H to replace $\mathcal{H}$. When H is a

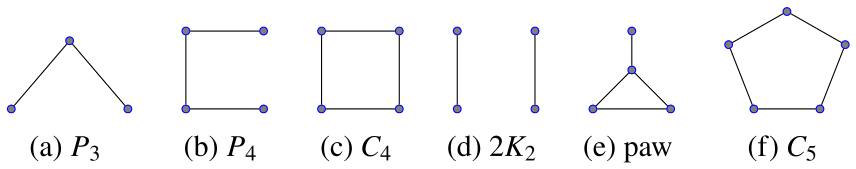


Figure 1.4: Some small graphs.

$\mathcal{H}$	completion	deletion	editing
$\{P_3\}$	trivial	$2k$	$2k$ [28]
$\{\text{paw}\}$	$O(k)$	$O(k^4)$	$O(k^6)$ [58]
$\{P_4, C_4\}$	$O(k^2)$	$O(k^3)$ [57]	$O(k^3)$ [57]
$\{2K_2, C_4, C_5\}$	$O(k^{1.5})$	$O(k^{1.5})$	–
$\{2K_2, C_4\}$	$O(k^{1.5})$	$O(k^{1.5})$	–

Table 1.1: The compressibility results we obtained for edge modification problems toward $\mathcal{H}$-free graphs. We use '–' to denote that the $\mathcal{H}$-free edge modification problem is in P.

graph with at most two vertices, the edge modification problems for H-free graphs are trivial. When H is a three-vertex graph, then $\{P_3\}$-free edge modification problems are the simplest of all nontrivial edge modification problems on H-free graphs. In Chapter 3, we study the kernelization algorithm for the $\{P_3\}$-free edge modification problem. We also call $\{P_3\}$-free graphs as cluster graphs, which is a disjoint union of cliques. The edge modification problems towards cluster graphs are well-studied. It is easy to see that the cluster completion problem can be solved in polynomial time (add edges to make every component of the input graph complete). There are several papers to show that the cluster editing problem is NP-hard [108, 99, 144]. The cluster editing problem is fixed-parameter tractable by Cai's result [23], and its FPT algorithm has been improved many times [79, 62, 132, 13]. For the kernelization algorithms for this problem, there are also numerous results [79, 133, 83, 13, 33, 28]. Cao and Chen [28] provided a $2k$-vertex kernel for the cluster editing problem. Their algorithm actually implies a $2k$-vertex kernel for the cluster deletion problem. We record this simple result for future reference. The cluster deletion problem is also studied widely and has been shown to be NP-complete in [122]. The parameterized complexity of this problem is studied in [79, 78, 45]. We study the kernelization algorithm for this problem. We show a kernelization algorithm for it that produces a $2k$-vertex kernel, improving on the $4k$-vertex kernel [81]. We also show that the same algorithm produces a kernel of the same size for the strong triadic closure problem, which, though originally not posed as an edge modification problem, is closely related to the cluster edge deletion problem [101].

We provide our results as follows.

Theorem 1.4.1. *The cluster deletion problem admits a $2k$-vertex kernel.*

Theorem 1.4.2. *The strong triadic closure problem admits a $2k$-vertex kernel.*

When H is a four-vertex graph, it is one of the graphs in Figure 1.5 (some four-vertex graphs are omitted because they are the complement of the ones presented here). We summarize the known results of kernelization algorithms for H-free edge modification problems in Table 1.2, where H is a four-vertex graph. The kernelization algorithms for edge modification problems toward claw-free graphs have been unknown to us until now.

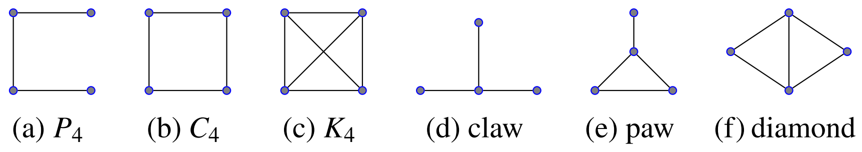


Figure 1.5: Graphs on four vertices.

H	completion	deletion	editing
K_4	trivial	$O(k^3)$	$O(k^3)$ [152]
P_4	$O(k^3)$	$O(k^3)$	$O(k^3)$ [82]
diamond	trivial	$O(k^3)$	$O(k^8)$ [31]
paw	$O(k)$ (our result)	$O(k^4)$ (our result)	$O(k^6)$ [58]
claw	unknown	unknown	unknown
C_4	no	no	no [82]

Table 1.2: The compressibility results of H-free edge modification problems for H being four-vertex graphs, where the kernels are measured by the number of vertices. Note that the results for the complement of H are implied; e.g., the answers are also no when H is $2K_2$, the complement of C_4.

In Chapter 4, we show polynomial kernels for H-free completion problem when H is a paw; see Figure 1.5(e). They answer open problems posed by Sandeep and Sivadasan [141].

Theorem 1.4.3. *The paw-free completion problem admits a* $38k$*-vertex kernel.*

When $\mathcal{H}$ contains several graphs, we consider the edge modification problems toward three well-studied $\mathcal{H}$-free graphs.

The first one is $\{P_4, C_4\}$-free graphs, also called trivially perfect graphs. All three edge modification problems (completion, edge deletion, editing) toward trivially perfect graphs are NP-complete [121, 145, 166] and are fixed-parameter tractable by Cai's result [23]. Drange et al. [56] showed that the trivially perfect editing problem could not be solved in subexponential time under the Exponential Time Hypothesis (ETH). For the edge deletion problem towards trivially perfect graphs, Liu et al. [112] provided the best-known FPT algorithm for it, whose running time is $O(2.42^k)$, and Drange et al. [55] showed that there is no parameterized subexponential algorithm for the problem under ETH. For the trivially perfect completion problem, Drange et al. [55] showed that it could be solved in parameterized subexponential time $2^{O(\sqrt{k}\log k)} \cdot n^{O(1)}$, and Bliznets et al. [12] showed that it could not be solved in time $2^{O(k^{1/4}/\log^c k)} \cdot n^{O(1)}$ under ETH, for some integer c. Regarding kernelization algorithms for the edge modification problems toward trivially perfect graphs, Drange et al. [56] gave an $O(k^7)$-vertex kernel for the trivially perfect editing problem, and Dumas et al. [57] improved the kernel to $O(k^3)$. Note that their algorithms imply the existence of kernels with the same size for the trivially perfect deletion problem and the trivially perfect completion problem. In Chapter 5, we study the trivially perfect completion problem. Guo [84] has claimed an $O(k^3)$-vertex kernel for it, but the details have never been published. We propose a very simple kernelization algorithm for the problem, which produces a kernel with at most $2k^2 + 2k$ vertices.

Theorem 1.4.4. *The trivially perfect completion problem admits a* $2k^2+2k$*-vertex kernel.*

Another one is $\{2K_2, C_4, P_5\}$-free graphs, also called split graphs. Split graphs also can be defined as graphs whose vertices can be partitioned into a clique and an independent set. By this definition, we can get that split graphs are closed under complementation, and the split completion problem and the split edge deletion problem are equivalent. Hammer and Simeone [89] showed that the split editing problem could be solved in polynomial time. We are concerned about the split edge deletion problem, which is shown to be NP-complete in [124]. Ghosh et al. [75] gave the first subexponential algorithm for it, whose running time is $2^{O(\sqrt{k}\log k)} \cdot n^{O(1)}$, and had been improved to $2^{O(\sqrt{k})}$ by Cygan et al. [40]. For the kernelization algorithm for this problem, the best-known result is an $O(k^2)$-vertex kernel [75].

In Chapter 6, we provide a kernel with $O(k^{1.5})$ vertices for it.

Theorem 1.4.5. *The split edge deletion problem has a kernel with $O(k^{1.5})$ vertices.*

Theorem 1.4.6. *The split completion problem admits a kernel with $O(k^{1.5})$ vertices and $O(k^{2.5})$ edges.*

The last one is $\{2K_2, C_4\}$-free graphs, also called pseudo-split graphs. A pseudo-split graph is either a split graph or a split graph plus a C_5 such that every vertex in the C_5 is adjacent to all vertices in the clique part of the split graph and nonadjacent to any vertex in the independent set part of the split graph. Also, pseudo-split graphs are closed under complementation. Drange [53] showed that the pseudo-split editing problem is solvable in polynomial time. For the convenience of presentation, we work on the edge deletion problem. The parameterized subexponential algorithm for it that runs in $2^{O(\sqrt{k}\log k)} \cdot n^{O(1)}$ time is given in [55]. In Chapter 6, we use the same algorithm for the split edge deletion problem to get a kernel with $O(k^{1.5})$ vertices for the pseudo-split completion problem.

Theorem 1.4.7. *The pseudo-split completion problem and the pseudo-split edge deletion problem have a kernel with $O(k^{1.5})$ vertices.*

Chapter 2

Preliminaries

In this chapter, we introduce all the notations and basic definitions that are used in this thesis.

2.1 Basic notions

All graphs discussed in this thesis are undirected and simple. A graph G is defined as a pair (V, E), $V(G)$ and $E(G)$ the vertex set and edge set of a graph G, respectively. Every edge in G is an unordered pair of vertices. We use uv to denote an edge $\{u, v\}$ and refer to u and v as the endpoints of the edge uv, and we call uv a non-edge if it is not an element of $E(G)$. A vertex v and an edge e are *incident* if v is one of the endpoints of e. Two vertices u and v are *adjacent* if $uv \in E(G)$. The sizes of the vertex set and the edge set are denoted by n and m, respectively.

For a vertex v in G, the *neighborhood* of v is denoted as $N_G(v)$, is the set of vertices adjacent to v, i.e., $N_G(v) = \{u \mid uv \in E(G)\}$. The *closed neighborhood* of v is denoted as $N_G[v]$, is $N_G(v)$ along with v itself, i.e., $N_G[v] = N_G(v) \cup \{v\}$. The *degree* of a vertex v in G, denoted as $d_G(v)$, is the number of vertices in $N_G(v)$. A vertex of degree 0 in a graph is called an *isolated vertex*. The *degree sequence* of a graph is a list of its vertex degrees, usually written in nonincreasing order. Given a vertex subset X of $V(G)$, the neighborhood of X, denoted as $N_G(X)$, is the set of vertices adjacent to any vertex in X, but not in X itself, i.e., $N_G(X) = \bigcup_{v \in X} N_G(v) \setminus X$. The closed neighborhood of X, denoted as $N_G[X]$, is $N_G(X)$ along with X itself, i.e., $N_G[X] = N_G(X) \cup X$. The degree of X, denoted as $d_G(X)$, is the number of vertices in $N_G(X)$, i.e., $d_G(X) = |N_G(X)|$. We write $d_G(v)$ instead of $d_G(\{v\})$ for a singleton set. We may omit the subscript when the context is clear.

A *clique* is a set of pairwise adjacent vertices, and an *independent set* is a set of pairwise nonadjacent vertices. For a positive integer k, a *k-partite graph* is a graph whose vertices can be partitioned into k different independent sets, called parts. A k-partite graph is complete if there is an edge between every pair of vertices from different parts. A *complete multipartite graph* is a graph that is complete k-partite (see an example in Figure 2.1).

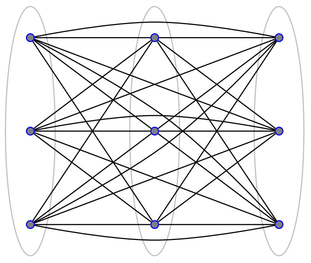


Figure 2.1: A complete multipartite graph.

A vertex v is *simplicial* if $N[v]$ is a clique, and a vertex v is universal if $N[v] = V(G)$. Two vertices u and v are *true twins* in G if their closed neighborhoods $N[u]$ and $N[v]$ are identical, and *false twins* if their neighborhoods $N(u)$ and

$N(v)$ are identical. A vertex set X is a *module* if every vertex in X has the same neighborhood outside X, i.e., $N(u) \setminus X = N(v) \setminus X$ for every pair of vertices u and v in X.

For two disjoint vertex subsets X and Y of $V(G)$, we say X is complete to Y if every vertex in X is adjacent to every vertex in Y, and use $E(X, Y)$ to denote the set of edges of which one endpoint is in X and the other in Y.

The *complement* of a graph G, denoted as $\overline{G}$, is a graph that has the same vertex set as G and two vertices u and u are adjacent in $\overline{G}$ if and only if they are nonadjacent in G. An *ℓ-vertex graph* is a graph with ℓ vertices. A graph G is *complete* if the vertex set of G forms a clique. We use K_ℓ to denote a complete graph with ℓ vertices and I_ℓ to denote an ℓ-vertex graph such that its vertex set is an independent set. A *subgraph* of a graph G is a graph H with vertex set $V(H)$ that is a subset of $V(G)$ and edge set $E(H)$ that is a subset of $E(G)$. If H is a subgraph of G, then we call G is a *supergraph* of H. An *induced subgraph* of G is a subgraph H such that every edge of G whose two endpoints are in $V(H)$ belongs to $E(H)$. If H is an induced subgraph of G with vertex set X, then we use $G[X]$ to denote H. For a subset $X \subseteq V(G)$, we denote by $G - X$ the subgraph $G[V(G) \setminus X]$, which is further shortened to $G - v$ when $X = \{v\}$.

A *walk* in a graph G of *length ℓ* is defined as an alternating sequence of vertices and edges, denoted as $\langle v_0, e_1, v_1, e_2, \ldots, e_\ell, v_\ell \rangle$, where $e_i = v_{i-1}v_i$ for $1 \le i \le \ell$. A *u-v walk* is a walk with the first vertex u and the last vertex v. A *path* is a walk with no repeated vertex, and a *trail* is a walk with no repeated edges. A *cycle* in a graph is a non-empty path in which only the first and last vertices are the same. The *length* of a path or a cycle is defined as the number of edges it contains. We use P_ℓ and C_ℓ to denote an induced path and an induced cycle on ℓ vertices, respectively.

A subgraph H of a graph is maximal, respectively minimal, with respect to some property if there is no proper supergraph, respectively proper subgraph, of H with that property. A graph G is *connected* if there exists at least one path connecting any two vertices in the graph. The *components* of a graph G are its maximal connected subgraphs. A component is trivial if it consists of a single vertex and is nontrivial otherwise.

A set X can be divided into subsets $X_1, X_2, \ldots, X_\ell$, forming a *partition* of X if the union of the subsets equals X and no two sets in the partition have any elements in common. A graph is *bipartite* if its vertex set can be partitioned into two independent sets, A and B. The vertices in A can be adjacent to some or all of the vertices in B. A *complete bipartite* graph is a special type of bipartite graph where every vertex in A is connected to all vertices in B, and it is denoted as $K_{p,\ell}$, where $|A| = p$ and $|B| = \ell$

We say that a vertex v and an edge e *dominate* each other if at least one of endpoints of e is adjacent to v. Note that an edge dominates, and is dominated by, both endpoints of this edge. A vertex v *dominates* a vertex set S if S is a subset of $N[v]$.

An *isomophism* from a graph G_1 to a graph G_2 is a bijection function $f : V(G_1) \to V(G_2)$ such that $uv \in E(G_1)$ if and only if $f(u)f(v) \in E(G_2)$. We say that a graph G_1 is *isomorphic* to a graph G_2 if there is an isomorphism from G_1 to G_2. A graph G *contains* a subgraph H if there is an induced subgraph of G that is isomorphic to H. For a set $\mathcal{H}$ of graphs, called *obstructions*, a graph G is *$\mathcal{H}$-free* if G does not contain any obstruction from $\mathcal{H}$. If $\mathcal{H}$ contains only one single graph H, we say that a graph G is *H-free* if G does not contain H. If every graph H in $\mathcal{H}$ is minimal, i.e., not containing any H' in $\mathcal{H}$ as a proper induced subgraph, then the set $\mathcal{H}$ of graphs are the minimal forbidden induced subgraphs of this graph class.

For a set F of non-edges, we denote by $G + F$ the graph with vertex set $V(G)$ and edge set $E(G) \cup F$. For a subset F of $E(G)$, we denote by $G - F$ the graph with vertex set $V(G)$ and edge set $E(G) \setminus F$. For $F \subseteq V(G) \times V(G)$, we denote by $G \triangle F$ the graph with vertex set $V(G)$ and edge set $(E(G) \setminus F) \cup (F \setminus E(G))$. For a vertex subset X of $V(G)$, we denote by $G - X$ the graph with vertex set $V(G) \setminus X$ and edge set $E(G) \setminus E(X, N[X])$. Let $\mathcal{G}$ be a graph class. The problems to be studied are formally defined as follows.

$\mathcal{G}$ Completion

Input: A graph G and a nonnegative integer k.

Output: Is there a set E_+ of at most k edges such that $G+E_+$ is in $\mathcal{G}$?

$\mathcal{G}$ Edge Deletion

Input: A graph G and a nonnegative integer k.

Output: Is there a set E_- of at most k edges such that $G - E_-$ is in $\mathcal{G}$?

$\mathcal{G}$ Editing

Input: A graph G and a nonnegative integer k.

Output: Is there a set $E_\pm$ of at most k edges such that $G \triangle E_\pm$ is in $\mathcal{G}$?

$\mathcal{G}$ Deletion

Input: A graph G and a nonnegative integer k.

Output: Is there a set V_- of at most k vertices such that $G - V_-$ is in $\mathcal{G}$?

Since it is always clear from the context what problem we are talking about, when we mention an instance (G, k), we do not always explicitly specify the problem. We use $\text{opt}(G)$ to denote the size of optimal solutions of G for the optimization version of a certain problem. Thus, an instance (G, k) is a yes-instance if and only if $\text{opt}(G) \leq k$.

For each problem, we apply a sequence of reduction rules. Each rule transforms an instance (G, k) to a new instance (G', k'). We say that a rule is safe if (G, k) is a yes-instance if and only if (G', k') is a yes-instance. Since all of our reduction rules are very simple and obviously doable in polynomial time, we omit the details of their implementation and the analysis of their running time.

2.2 Graph classes

We first introduce some general notations and then describe the graph classes we studied in this thesis. A property is said to be non-trivial if it holds for at least one graph, but not for all graphs. If a property is preserved under vertex deletion, it is called hereditary. A graph class is hereditary if given a graph G in the class, then every induced subgraph of G is also in the class. A large number of graph classes we have learned are hereditary, such as forests, bipartite graphs, planar graphs, perfect graphs, and chordal graphs. Let $\mathcal{H}$ be a set of graphs. We use $\mathcal{G}_{\mathcal{H}}$

to denote the graph class of all $\mathcal{H}$-free graphs and say that $\mathcal{G}_{\mathcal{H}}$ is characterized by $\mathcal{H}$. Cai [23] has shown another way to look at hereditary graph classes.

Theorem 2.2.1 ([23]). *A graph class $\mathcal{G}$ is hereditary if and only if it can be characterized by a (possible infinite) set of graphs.*

In our work, we only study some hereditary graph classes $\mathcal{G}_{\mathcal{H}}$ where $\mathcal{H}$ is a finite set of graphs. We partition them into two parts: $\mathcal{H}$ consist of only one graph or not. When $\mathcal{H}$ consists of one graph $\{H\}$, we write it as H.

2.2.1 H-free graphs

For H-free graphs, we are interested in H-free edge modification problems when H is a small graph, and we list all small graphs with at most four vertices in Figure 2.2.

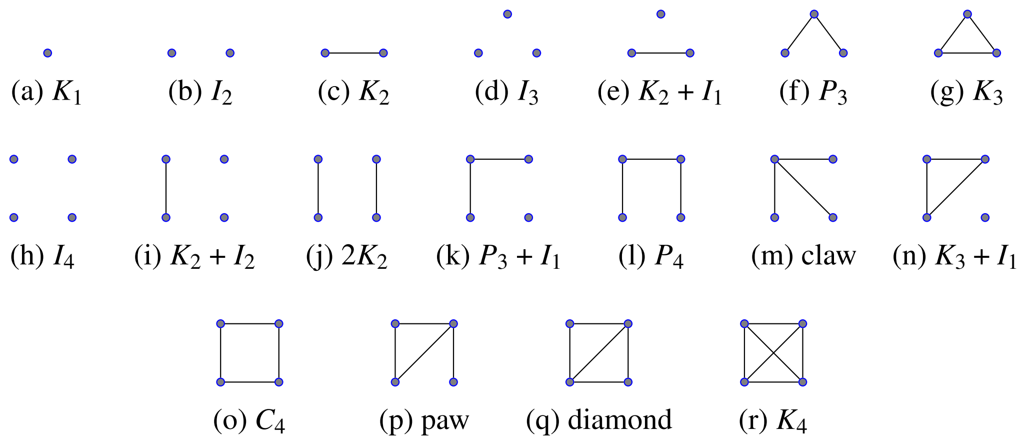


Figure 2.2: Graphs with at most four vertices.

We do not consider edge modification problems toward H-free graphs when H is a one-vertex graph, which is K_1, because we cannot get a K_1-free graph by modifying edges on a given graph. There are two two-vertex graphs (see Figure 2.2(b)(c)). Since the H-free completion problem is equivalent to the $\overline{H}$-free edge deletion problem and $K_2 = \overline{I_2}$, we only need to consider the edge modification problems toward K_2-free graphs. Then the edge modification problems for K_2-free graphs can be solved by deleting all edges in polynomial time. When H is a three-vertex graph, there are four candidates for it (see Figure 2.2(d)–(g)). Since $K_3 = \overline{I_3}$ and $P_3 = \overline{K_2 + I_1}$, we only need to consider the edge modification problems towards K_3-free graphs and P_3-free graphs. In this thesis, we study the P_3-free graphs, also called *cluster graphs*.

The cluster graphs is a simple and vital graph class that has been studied widely. A graph is a cluster graph if the vertex set of it can be partitioned into a set of disjoint cliques. The complement of a cluster graph isz a complete multipartite graph or a 2-leaf power [128].

When H is a four-vertex graph, there are eleven graphs (see Figure 2.2(h)–(r)). For H-free edge modification problems, we only need to consider six four-vertex graphs (since we do not consider the complement of a graph). We can see that $I_4 = \overline{K_4}$, diamond= $\overline{K_2 + I_2}$, $C_4 = \overline{2K_2}$, paw=$\overline{P_3 + I_1}$, claw= $\overline{K_3 + I_1}$, and $P_4 = \overline{P_4}$.

We study paw-free graphs here.

Proposition 2.2.2 ([129]). *A graph is paw-free if and only if every component of it is triangle-free or a complete multipartite graph with at least three parts.*

In other words, if a connected paw-free graph contains a triangle, then it is necessarily a complete multipartite graph.

2.2.2 $\mathcal{H}$-free graphs

When $\mathcal{H}$ contains more than one graph, we study several $\mathcal{H}$-free graphs where $\mathcal{H}$ is a set of small graphs with at most five vertices.

The first one is $\{P_4, C_4\}$-free graphs, also called *trivially perfect graphs* [77]. A graph is a trivially perfect graph if, for every induced subgraph, the size of the maximum independent set equals the number of maximal cliques. Trivially perfect graphs are also known as comparability graphs of trees [161], quasi-threshold graphs [165], or intersection graphs of nested intervals [19], here a set of nested intervals is a set of intervals such that every two intervals in the set either are disjoint or one contains the other (see Figure 2.3).

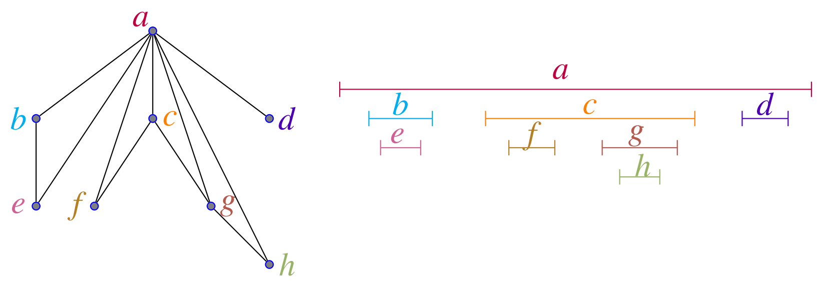


Figure 2.3: An example to show a trivially perfect graph is an intersection graph of nested intervals.

Another $\mathcal{H}$-free graph we studied is $\{2K_2, C_4, C_5\}$-free graphs, which is also called *split graphs* [64]. A graph is a split graph if its vertex set can be partitioned into a clique and an independent set. We use $C \uplus I$, where C being a clique and I an independent set, to denote a split partition of a split graph. Note that a split graph may have more than one split partition; e.g, a complete graph K_n has $n+1$ different split partitions. The class of split graphs has many characterizations, and some of them are introduced here. Foldes and Hammer [64] showed that a graph G is a split graph if and only if G and $\overline{G}$ are chordal graphs. Brandstädt et al. [19] showed that a graph is a split graph if and only if G is an intersection graph of a set of distinct substars of a star, where a star is a bipartite graph such that one side of its partition has only one vertex. The degree-sequence characterization is one of the most nice characterizations of split graphs [89, 155]. A graph G with the degree sequence $d_1 \geq d_2 \geq \ldots \geq d_n$ is a split graph if and only if $\sum_{i=1}^{j} d_i = j(j-1) + \sum_{i=j+1}^{n} d_i$, where j is the largest index in the degree sequence such that $d_j \geq j - 1$. From the definition, we can see that the complement of a split graph is also a split graph. Thus, the split completion problem is polynomially equivalent to the split edge deletion problem.

The last $\mathcal{H}$-free graph class we studied is $\{2K_2, P_4\}$-free graphs, also called *pseudo-split graphs* [115]. The class of pseudo-split graphs is a superclass of split graphs. In particular, split graphs are precisely C_5-free pseudo-split graphs. A pseudo-split graph is a graph whose vertex set can be partitioned into a clique C, an independent set I, and a set S such that (1) S induces a C_5; (2) $C \subseteq N(v)$ for every $v \in S$; and (3) $I \cap N(v) = \emptyset$ for every $v \in S$, where S may or may not be empty. We say that $C \uplus I \uplus S$ is a pseudo-split partition of the graph, where S may or may not be empty. If S is empty, then the graph is a split graph and $C \uplus I$ is a split partition of it. Otherwise, the graph has a unique pseudo-split partition. (One may also verify that S is a module.) Similar as split graphs, the complement of a pseudo-split graph remains a pseudo-split graph. Thus, the completion problem and the edge deletion problem toward pseudo-split graphs are polynomially equivalent. Note that a pseudo-split graph contains at most one C_5. In case that a pseudo-split graph does contain a C_5, removing any vertex from the C_5 leaves a split graph. Therefore, those two classes are very "close."

2.2.3 Other classes

There are several graph classes that will be mentioned in this thesis and are closely related to the graph classes we introduced above.

A *chordal graph* is a graph that every cycle of length larger than three in it has a chord. The forbidden induced subgraphs of chordal graphs are all induced cycles with lengths larger than three. From the forbidden induced subgraphs of chordal graphs and split graphs (every induced cycle with a length larger than five contains a $2K_2$), we can easily get that split graphs is a subclass of chordal graphs. A graph is an *interval graph* if its vertices can be assigned to intervals on a real line such that there is an edge between two vertices if and only if their corresponding intervals intersect. A graph is a *unit interval graph* if it is an interval graph and all the intervals have the same length. We can get that interval graphs is a subclass of chordal graphs from another definition of chordal graphs and interval graphs, i.e., chordal graphs are intersection graphs of subtrees of a tree, and interval graphs are intersection graphs of sub-paths of a path. Lekkerkerker and Boland [109] showed that a graph is an interval graph if and only if it is chordal and contains no asteroidal triple (Three independent vertices construct an asteroidal triple if there exists a path between each pair of them avoiding every neighbor of the third one).We get that trivially perfect graphs is a subclass of interval graphs since a trivially perfect graph can be represented as a set of nested intervals. A graph is a *complete split graph* if every vertex in the clique side is adjacent to all vertices in the independent set side of a split partition. A graph G is a *threshold graph* if there is a real number t (called *threshold*) and an assignment $f : V(G) \to \mathbb{R}$ such that uv is an edge in $E(G)$ if and only if $f(u) + f(v) \geq t$ [36]. The forbidden induced subgraphs of threshold graphs are $2K_2$, C_4, and P_4 [37]. For a complete split graph G with a split partition $C \uplus I$, we can get that G is a threshold graph by assigning $f(v) = t$ for every vertex v in C and $f(u) = 0$ for every vertex in I. Therefore, complete split graphs is a subclass of split graphs. For a threshold graph G with threshold t, we can partition every vertex v with $f(v)$ not less than $\frac{t}{2}$ into a set and other vertices into another set, then we can get a threshold partition. Hence, threshold graphs is a subclass of split graphs. For a threshold graph G with a split partition $C \uplus I$, where $C = \{v_1, v_2, \ldots, v_{|C|}\}$ and $I = \{u_1, u_2, \ldots, u_{|I|}\}$. If we order the vertices in $V(G)$ such that $f(u_1) \leq f(u_2) \leq \ldots \leq f(u_{|I|}) < \frac{t}{2}$ and $\frac{t}{2} \leq f(v_1) \leq f(v_2) \leq \ldots \leq f(v_{|C|})$, then we can get the inclusion relation of neighbors of vertices in the clique part and the independent set part that $N(u_1) \subseteq N(u_2) \subseteq \ldots \subseteq N(u_{|I|})$ and $N(v_1) \subseteq N(v_2) \subseteq \ldots \subseteq N(v_{|C|})$.

Recall that a bipartite graph is a graph whose vertices can be partitioned into two independent sets. The relationship between split graphs and bipartite graphs is that we can obtain a split graph from a bipartite graph by adding edges to make one side of the bipartite graph into a clique. A similar relationship also exists between threshold graphs and chain graphs, i.e., we can get a chain graph by deleting all edges in the clique side of a threshold graph [168, 166].

2.2.4 Graph modification problems on (subclasses of) chordal graphs

As stated before, most graph modification problems are NP-complete, where the input graphs are general graphs. There are several things we could do for NP-complete problems if we do not want to give up. One is to try to find a solution that is good enough but not optimal, which means finding an approximation algorithm [159]. One way is to design better exact algorithms for them [6, 160, 66, 65], which is to design an algorithm to obtain exact solutions with good exponential running times. Restricting the input graph classes to smaller classes is also an approach we could use to deal with those NP-complete problems [150, 95, 74, 42, 30]. Another classical way to deal with NP-complete problems is to design parameterized algorithms [47, 48]. In this thesis, we will study the last two methods.

For an NP-complete graph modification problem, restricting the input graph as a special graph is also a way to understand the hardness of the problem more deeply. We have studied some vertex deletion problems such that both the input graph and the target graph are a graph in subclasses of chordal graphs [30], where those problems are NP-complete when the input graph is a general graph. We get that some of them are still NP-complete, and some of them become polynomial-time solvable. Figure 2.4 shows the results and major graph classes studied in [30] and the relationship among these graph classes. In Figure 2.4, a solid edge between two classes means that the lower one is a subclass of the higher one, and a directed dashed edge from $\mathcal{G}_1$ to $\mathcal{G}_2$ is used when $\mathcal{G}_2$ is not an immediate subclass of $\mathcal{G}_1$. The cyan, violet, and black edges indicate that the complexity of the representing problems is in P, NP-complete, and unknown, respectively.

As we can see, some of these problems are still NP-complete. We first recall several results from [30] here.

Theorem 2.2.3. *The vertex deletion problem from split graphs to threshold graphs is NP-complete.*

Since split graphs and threshold graphs can be recognized in polynomial time [76],

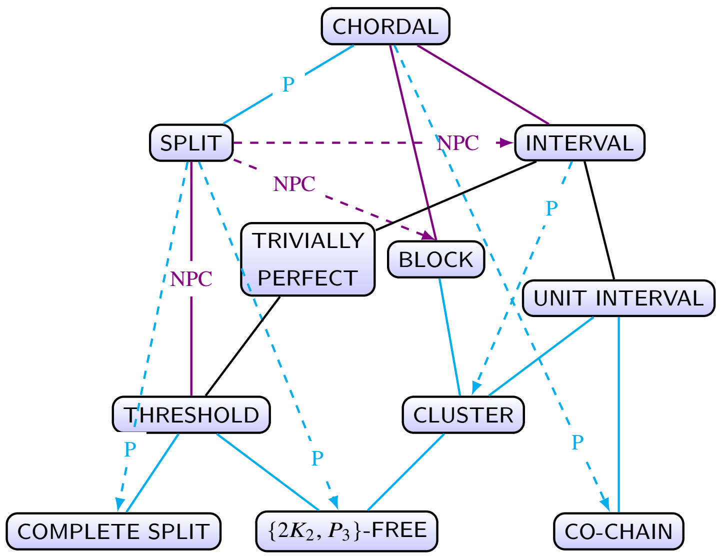


Figure 2.4: A summary of results and major graph classes studied in [30].

the vertex deletion problem from split graphs to threshold graphs is in NP. Now we prove it is NP-hard by using a reduction in [168] that is used to prove the NP-hardness of the vertex deletion problem from bipartite graphs to chain graphs. Let G be a bipartite graph with partition C and I. By adding all possible edges among C to make it a clique, we can get a split graph with split partition $C \uplus I$, and we use G' to denote it. We claim that for every vertex set X that $G[X]$ is a chain graph, i.e., being $2K_2$-free, if and only if $G'[X]$ is a threshold graph, i.e., being P_4-free. Let X be a set of four vertices. If $G[X]$ is a $2K_2$, then we can get that $|X \cap C| = |X \cap I| = 2$, then $G'[X]$ is a P_4. Similarly, if $G'[X]$ is a P_4, then we can get that $G[X]$ is a $2K_2$ (since $|X \cap C| = |X \cap I| = 2$). Therefore, we can get that this problem is NP-complete.

We also proved that the vertex deletex problem from split graphs to interval graphs is NP-complete. Here we use threshold graphs, which is a commom subclass of split graphs and interval graphs, to help us to complete our proof. Recall that threshold graphs is a subclass of interval graphs, and this can be generalized as follows. Let G_1 and G_2 be two threshold graphs with split partitions $C \uplus I$ and $C' \uplus I'$, respectively. We use $G_1 \bowtie_{(C,C')} G_2$, or simply $G_1 \bowtie G'$ in the rest of the paper if it does not cause any ambiguity, denote the graph obtained from G_1 and G_2 by adding all possible edges between C and C'—i.e., its vertex set and edge set are $V(G_1) \cup V(G_2)$ and $E(G_1) \cup E(G_2) \cup (C \times C')$ respectively. This is clearly a split graph with the split partition $C \cup C'$ and $I \cup I'$. See an example in Figure 2.5. One can verify that $G_1 \bowtie G_2$ is also an interval graph by their obstructions as follows. A split graph that is not an interval graph has to contain a tent, a net, or a rising sun (see Figure 2.6). Each of them has three independent vertices, which have to be from $I \cup I'$. Suppose that there is a net $\{a, b, c, x, y, z\}$ in $G_1 \bowtie G_2$, where a, b, c are three independent vertices in it and x, y, z are neighbors of a, b, c, respectively. At least two of $\{x, y, z\}$ come from same graph (G_1 or G_2), suppose x and y are from G_1. Then we can see that the subgraph induced by $\{a, b, x, y\}$ is not a threshold graph, which contradict that G_1 is a threshold graph. All of these three graphs cannot be contained in $G_1 \bowtie G_2$ by using similar observation.

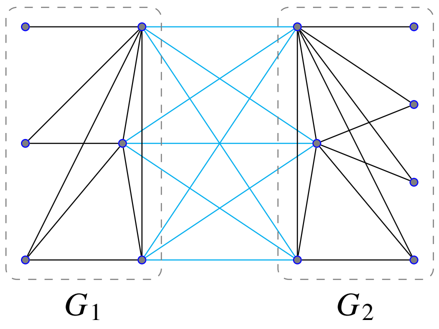
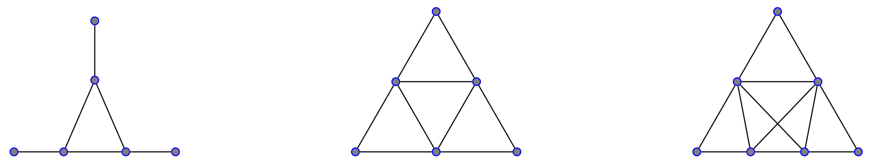


Figure 2.5: An example for the graph $G_1 \bowtie G_2$.

Proposition 2.2.4. *For any two threshold graphs G_1, G_2, the graph $G_1 \bowtie G_2$ is an interval graph.*



(a) net (b) tent (c) rising sun

Figure 2.6: Minimal split graphs that are not interval graphs.

A better way to look at Proposition 2.2.4 is probably through interval models. Recall that an interval model for an interval graph is a set of intervals representing its vertices. In this thesis, all intervals are closed. An interval model can be specified by the $2n$ endpoints for the n intervals. We use $[\mathtt{lp}(v), \mathtt{rp}(v)]$ to denote an interval for vertex v.

Let G be a threshold graph with split partition $C \uplus I$, and let vertices in I be ordered in a way that $N(v_1) \subseteq N(v_2) \subseteq \cdots \subseteq N(v_{|I|})$. We can build an interval model for G by setting intervals

$$\begin{array}{rr} [i, i+0.5] & \text{for every } v_i \in I, \\ \big[\min\{i : v_i \in N(v)\}, |I|+2\big] & \text{for every } v \in N(I), \text{ and} \\ \big[|I|+1, |I|+2\big] & \text{otherwise } (i.e., v \in C \setminus N(I)). \end{array} \tag{2.1}$$

See Figure 2.7 for illustration.

An interval model for $G_1 \bowtie G_2$ can be built from the interval models for G_1 and G_2 by (i) keeping the intervals for G_1, and (ii) setting the interval to be $\big[|I| + |I'| + 3 - \mathtt{rp}(v), |I| + |I'| + 3 - \mathtt{lp}(v)\big]$ for each $v \in V(G_2)$. See Figure 2.8 for illustration.

Theorem 2.2.5. *The vertex deletion problem from split graphs to interval graphs is NP-complete.*

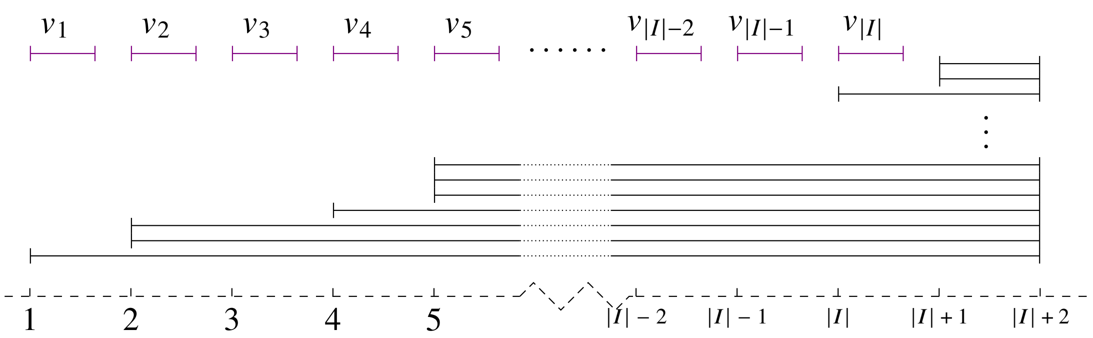


Figure 2.7: The interval model for a threshold graph given by (2.1).

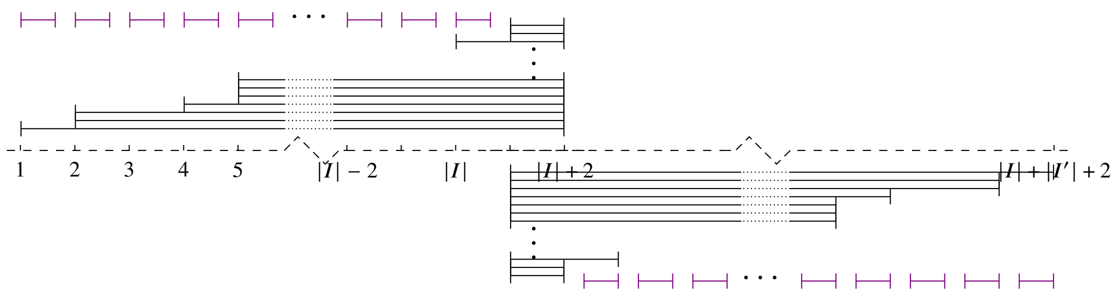


Figure 2.8: The interval model for $G_1 \bowtie G_2$.

Since split graphs and interval graphs can be recognized in polynomial time [76, 18, 102, 93, 86, 38, 27], the vertex deletion problem from split graphs to interval graphs is in NP. Let G be a split graph with split partition $C \uplus I$. We take a complete split graph G' with split partition $C' \uplus I'$, where $|C'| = |I'| = |C|$, and let $H = G \bowtie G'$. We argue that (G, k) is a yes-instance of the vertex deletion problem from split graphs to threshold graphs if and only if (H, k) is a yes-instance of the vertex deletion problem from split graphs to interval graphs. When $k \geq |C|$, it is trivial to get that these two problems are yes-instance. Now we assume that $k < |C|$.

Suppose that a vertex set V_- is a solution of (G, k), then $G - V_-$ is a threshold graph. According to Proposition 2.2.4, the graph $(G - V_-) \bowtie G'$ is an interval graph. It is the same graph as $H - V_-$. Therefore, the solution V_- is also a solution of (H, k). This verifies the only if direction.

Now suppose that $H - V_-$, where $|V_-| \leq k$, is an interval graph. Let $G'' = G - \big(V_- \cap V(G)\big)$. Suppose for contradiction that G'' is not a threshold graph, then G'' must contain some P_4. Assume that $v_1u_1u_2v_2$ is a P_4 in G''. Since G'' is a split graph, we must have u_1, and u_2 are vertices in C and v_1, and v_2 are vertices in I. On the other hand, by the assumption $k < |C|$, neither $C' \setminus V_-$ nor $I' \setminus V_-$ can be empty. Let $u \in C' \setminus V_-$ and $v \in I' \setminus V_-$. By the construction, the only edges between $\{u, v\}$ and $\{v_1, u_1, u_2, v_2\}$ are uu_1 and uu_2, but then these six vertices together induce a net in $H - V_-$, which is a contradiction. Then by Theorem 2.2.3, we can ge that the vertex deletion problem from split graphs to interval graphs is NP-complete.

We also find that several graph modification problems become easier when the input graph is a graph in a subclass of chordal graphs, which means these problems could be solved in polynomial time. Next we recall some polynomial-time algorithms from [30]. Our focus would be laid on the use of structural properties, and if possible, we would present the simplest algorithms without explaining on the implementation details. These problems may have more efficient algorithms, and with more complex data structures and algorithmic finesses, some of them may even be solved in linear time.

We present two algorithms on split graphs in detail, for which we need to put split partitions under scrutiny. It is a trouble to us that a split graph can have many different partitions. Hence the first thing we need to do is to restrict split partitions that a split graph could have. Let $C \uplus I$ be a split partition of a split graph G. If some vertex in I is completely adjacent to C, then we can move such a vertex v to C to make another split partition $C' = C \cup \{v\}$ and $I' = I \setminus \{v\}$. Note that the vertex v may not be unique, and the resulting graphs by moving them would be isomorphic.

Moreover, after such a move, no vertex of I' can be completely adjacent to C'. The following proposition fully characterizes split graphs with more than one different split partition.

Proposition 2.2.6. *Let G be a split graph with at least two split partitions, and let $C \uplus I$ and $C' \uplus I'$ be two different split partitions of G.*

(i) The difference between $|C|$ and $|C'|$ is at most one.

(ii) If $|C| = |C'| + 1$, then C is a maximum clique, and I' is a maximum independent set of G; moreover, $C' \subset C$.

(iii) If $|C| = |C'|$, then $G - E(C)$ and $G - E(C')$ are isomorphic.

As a result, a split graph has either one or two essentially distinct split partitions. On the other hand, of all split partitions of a complete bipartite graph, only one, whose independent set is the largest, satisfies the definition of complete bipartite graphs, and we will exclusively refer to it when we are discussing a complete split graph.

Let G be a split graph with split partition $C \uplus I$ and let G' be a $\{2K_2, P_3\}$-free subgraph of G. Note that $\{2K_2, P_3\}$-free graphs are cluster graphs with only one nontrivial clique. If G' has edges, all of them must be in the same nontrivial clique. At most one vertex of this clique can be from I; therefore, all other vertices of I either are deleted or become isolated in G'. In other words, for each other vertex v in I, either v or all its neighbors have to be deleted.

Theorem 2.2.7. *The vertex deletion problem from split graphs to $\{2K_2, P_3\}$-free graphs is in P.*

```
0.  $\mathcal{S} \leftarrow \emptyset$;
1.  build a bipartite graph $G'$ by removing all edges among $C$ from $G$;
2.  find a minimum vertex cover of $G'$, and add it to $\mathcal{S}$;
3.  **for each** $v \in I$ **do**
        find a minimum vertex cover $X$ of $G' - (C \setminus N(v)) - v$;
        add $X \cup (C \setminus N(v))$ to $\mathcal{S}$;
4.  **return** a set in $\mathcal{S}$ with the minimum cardinality.
```

Figure 2.9: An algorithm for the vertex deletion problem from split graphs to $\{2K_2, P_3\}$-free graphs.

Let G be the input graph to the vertex deletion problem from split graphs to $\{2K_2, P_3\}$-free graphs. Let $C \uplus I$ be a split partition of G. We use the algorithm in Figure 2.9 to find a minimum solution to G. To argue its correctness, we show that (i) every set in $\mathcal{S}$, added in step 2 or 3, is a solution to G, and (ii) at least one of them is minimum. For (i), it is easy to verify that any vertex cover of $G' = G - E(C)$ is a solution: There is no edge between C and I after its deletion. The situation in step 3 is similar; note that $N[v] \uplus (I \setminus \{v\})$ is a split partition of $G - \big(C \setminus N(v)\big)$.

Let V_- be a minimum solution to G. In the first case, every vertex $v \in I \setminus V_-$ is isolated in $G - V_-$. In other words, V_- contains a vertex cover of $G' = G - E(C)$, and then the solution found by step 2 is already the minimum. Henceforth we assume that there exists a vertex $v \in I \setminus V_-$ such that $N(v) \not\subseteq V_-$. Since any vertex $u \in N(v)$ and $w \in C \setminus N(v)$ induce a P_3 with v, in this case all vertices in $C \setminus N(v)$ must be in V_-. Note that the vertex v is unique: If two vertices in $I \setminus V_-$ have neighbors in $C \setminus V_-$, then they are in a non-clique component. Therefore, after removing $C \setminus N(v)$ and v from the graph, it reduces to the first case. This justifies step 3.

The algorithm in Figure 2.9 makes $O(n)$ calls to an algorithm for the bipartite vertex cover problem, each taking $O(m\sqrt{n})$ time, and hence the whole algorithm runs in $O(mn\sqrt{n})$ time. Therefore, we get that the vertex deletion problem from split graphs to $\{2K_2, P_3\}$-free graphs is in P.

Since $\{2K_2, P_3\}$-free graphs is a common subclass of split graphs and cluster graphs, we can apply the algorithm of Theorem 2.2.7 to the vertex deletion problem from split graphs to cluster graphs. Moreover, since split graphs is self-complementary, while the complement a $\{2K_2, P_3\}$-free graph is a complete split graph, we get that the vertex deletion problem from split graphs to complete split graphs is also in P.

Corollary 2.2.8. *The vertex deletion problem from split graphs to cluster graphs and the vertex deletion problem from split graphs to complete split graphs are in P.*

A similar observation can be used to solve the vertex deletion problem from split graphs to unit interval graphs. We start from a simple property of connected graphs in a common subclass of split graphs and unit interval graphs. Figure 2.10 shows a unit interval model of a connected split graph with a split partition $C \uplus I$ that is also a unit interval graph, where violet intervals are for vertices in C and cyan for I. Note that the vertex v from I is completely adjacent to C.

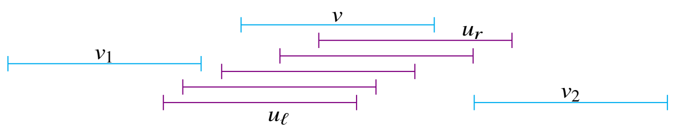


Figure 2.10: An interval model for a connected split graph that is also a unit interval graph.

Proposition 2.2.9. *Let G be a connected split graph and let $C \uplus I$ be a split partition of G. If G is a unit interval graph, then $|I| \leq 3$, and the equality holds only when there is a vertex $v \in I$ adjacent to all vertices in C.*

Proof. We prove $|I| \leq 2$ if C is a maximum clique of G, and then the proposition follows from Proposition 2.2.6(i). Let u_ℓ and u_r be the vertices in C with respectively the leftmost and rightmost intervals. Suppose for contradiction that $|I| > 2$. Let v_1 and v_2 be the vertices in I with respectively the leftmost and rightmost intervals. Then $\mathtt{lp}(u_\ell) < \mathtt{rp}(v_1) < \mathtt{lp}(u_r) < \mathtt{rp}(u_\ell) < \mathtt{lp}(v_2) < \mathtt{rp}(u_r)$, where the second and the fourth inequalities follow from that C is a maximum clique, and the others from the selections of the four vertices. Since G is connected, the interval for any other vertex v in $I \setminus \{v_1, v_2\}$, which is nonempty, has to lie in between $\mathtt{rp}(v_1)$ and $\mathtt{lp}(v_2)$. But then it has to contain $[\mathtt{lp}(u_r), \mathtt{rp}(u_\ell)]$, and $C \cup \{v\}$ is a clique, contradicting that C is a maximum clique of G. □

Similar as Theorem 2.2.7, our algorithm for the vertex deletion problem from split graphs to unit interval graphs separates into two cases, based on whether there is a vertex in $I \setminus V_-$ adjacent to all vertices in $C \setminus V_-$.

Theorem 2.2.10. *The vertex deletion problem from split graphs to unit interval graphs is in P.*

Let G be the input graph to the vertex deletion problem from split graphs to unit interval graphs and let $C \uplus I$ be a split partition of G. We use the algorithm in Figure 2.11 to find a solution. To argue its correctness, we show that all sets put into $\mathcal{S}$ in steps 1–4 are solutions to G, and at least one of them is minimum. It is clear for step 1. After the deletion of a solution found in step 2, only the vertex v in I remains adjacent to the remaining vertices of C. In step 3, only the vertices v_1 and v_2 from I can remain adjacent to vertices in C. In step 3.2, no vertex in C is adjacent to both v_1 and v_2; in step 3.3, every vertex in C is adjacent to at least one of v_1 and v_2. In either case, it is easy to verify that the graph is a unit interval graph

```
0.   $\mathcal{S} \leftarrow \emptyset$;
1.   solve the vertex deletion problem from split graphs to $\{2K_2, P_3\}$-free
     graphs on $G$, and add the solution to $\mathcal{S}$;
     case 1:
2.   for each $v \in I$ do
         find a minimum vertex cover of $G - v - E(C)$, and add it to $\mathcal{S}$;
3.   for each $v_1, v_2 \in I$ do
3.1.     $G' \leftarrow G - \{v_1, v_2\} - E(C)$;
3.2.     find a minimum vertex cover of $G' - N(v_1) \cap N(v_2)$,
         and add its union with $N(v_1) \cap N(v_2)$ to $\mathcal{S}$;
3.3.     find a minimum vertex cover of $G' - C \setminus \big(N(v_1) \cup N(v_2)\big)$,
         and add its union with $C \setminus \big(N(v_1) \cup N(v_2)\big)$ to $\mathcal{S}$;
     case 2:
4.   for each $v \in I$ do
         $G'' \leftarrow G - \big(C \setminus N(v)\big)$ with split partition $N[v]$ and $I \setminus \{v\}$;
         solve $G''$ as case 1, but append $C \setminus N(v)$ to each solution found;
5.   return a set in $\mathcal{S}$ with the minimum cardinality.
```

Figure 2.11: An algorithm for the vertex deletion problem from split graphs to unit interval graphs.

by building a unit interval model directly. Step 4 follows from the same argument as above: After the deletion of $C \setminus N(v)$, it reduces to one of the three previous steps.

Let V_- be a minimum solution to G. If $G - V_-$ is a $\{2K_2, P_3\}$-free graph, then the solution found by step 1 is the minimum. Henceforth we assume that $G - V_-$ contains a component $G[U]$ such that U is not a clique; note that such a component contains all vertices in $C \setminus V_-$ and hence is unique.

In the first case, every vertex $v \in U \cap I$ has at least one non-neighbor in $C \setminus V_-$, i.e., $N(v) \setminus V_- \subset C \setminus V_-$. According to Proposition 2.2.9, we have $|U \cap I| \leq 2$. If $U \cap I = \{v\}$, then $G - (V_- \cup \{v\})$ is a $\{2K_2, P_3\}$-free graph and the only nontrivial clique $U \setminus \{v\}$ is a subset of C; hence step 2 always find a minimum solution. In the rest of this case, the set $U \cap I$ has two different vertices; let them be v_1 and v_2.

Since any $u_1 \in N(v_1) \cap N(v_2)$ and $u_2 \in C \setminus \big(N(v_1) \cup N(v_2)\big)$ induce a claw with $\{v_1, v_2\}$, at least one of the two sets needs to be empty or completely contained in V_-. Steps 3.2 and 3.3 take care of these two situations separately.

We are now in the second case, where $C \setminus V_- \subseteq N(v)$ for some vertex $v \in I \setminus V_-$; in other words, V_- contains all vertices in $C \setminus N(v)$. There might be two of such vertices, when we can take v to be either of them. Clearly, $N[v]$ and $I \setminus \{v\}$ is then a split partition of $G'' = G - \big(C \setminus N(v)\big)$, which has a solution $V_- \setminus \big(C \setminus N(v)\big)$. Moreover, under this new split partition, we reduce it to the first case.

The algorithm in Figure 2.11 makes $O(n^3)$ calls to the algorithm for the bipartite vertex cover problem, each taking $O(m\sqrt{n})$ time, and hence the whole algorithm runs in $O(mn^{3.5})$ time. Therefore, we have that the vertex deletion problem from split graphs to unit interval graphs is in P.

2.3 Parameterized algorithms

Parameterized complexity is a framework in computer science that aims to analyze the computational complexity of problems beyond the traditional polynomial time complexity analysis. Downey and Fellows [47, 48, 49, 50, 51] initially studied the theory of parameterized complexity, and they gave a systematic complexity analysis of parameterized problems. In parameterized complexity, we want to find a parameter that really affects the complexity of the problem. Note that there may be several choices of parameters for the same problem.

A *parameterized problem* is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a fixed, finite alphabet. An input of a parameterized problem is an instance $(x, k) \in \Sigma^* \times \mathbb{N}$, where k is called the parameter. A parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$ is *fixed-*

parameter tractable (FPT) if given an instance (x, k) of it there is an algorithm that can correctly determine whether (x, k) is in L in time $f(k) \cdot |(x, k)|^c$, where $f : \mathbb{N} \to \mathbb{N}$ is a computable function and c is a constant.

2.4 Kernelization algorithms

A *kernelization algorithm*, or in short, *kernelization* is a polynomial-time algorithm that takes an instance (x, k) of a parameterized problem as input and produces an equivalent instance (x', k') such that (x, k) is a yes-instance if and only if (x', k') is a yes-instance, where $k' \leq k$ and the size of G' is bounded by a computable function of k'. The output instance (x', k') is a *polynomial kernel* if the size of G' is bounded from above by a polynomial function of k'.

It is known that a parameterized problem is fixed-parameter tractable if and only if it admits a kernelization. If a parameterized problem admits a kernelization, we can apply a polynomial-time kernelization algorithm to every instance of the problem to obtain an equivalent instance whose size is bounded by a function of the parameter. We can then use an algorithm to determine whether the resulting instance is a yes-instance, and the running time depends only on the parameter. This enables us to quickly establish that the problem is fixed-parameter tractable. For the other direction, it is not immediately apparent but not difficult to see, as demonstrated in [67].

We will focus on the kernelization of edge modification problems in this thesis. Many techniques have been used to design a kernelization algorithm for an FPT problem, as shown in [114, 68, 103, 67]. These techniques include crown decomposition [35, 113], linear programming [125, 32], sunflower lemma [59], modular decomposition [9, 82], matroids [105, 106], and rainbow matching [8]. In this chapter, we introduce the most intuitive method used in kernelization algorithms for both vertex modification problems and edge modification problems in a hereditary graph class $\mathcal{G}$ that is characterized by finite obstructions.

2.4.1 The sunflower lemma for vertex modification

Let $\mathcal{G}$ be a hereditary graph class with finite obstructions. The sunflower lemma was first introduced by Erdös and Rado [59], which is a most intuitive and simple method used in kernelization algorithms for vertex modification problems. Let us first introduce the definition of the sunflower. A sunflower is a collection of k sets $S_1, S_2, \ldots, S_k$ such that the intersection of any pair of distinct sets is identical (use Y to denote it) and every set minus the intersection is nonempty, we call the intersection as the core of the sunflower and $S_i \setminus Y$ for every $1 \leq i \leq k$ as a petal of the sunflower.

Theorem 2.4.1 ([59]). *Let $\mathcal{S}$ be a family of distinct sets over a universe U, and each set has cardinality at most d. If $|\mathcal{S}| > d!(k-1)^d$, then $\mathcal{S}$ contains a sunflower with k petals.*

For the vertex deletion problem to $\mathcal{G}$, we need to hit all of the obstructions in the input graph. Then we can reduce the vertex deletion problem to the d-hitting set problem, where d is the size of an obstruction for $\mathcal{G}$ with a maximum number of vertices. By Theorem 2.4.1, we can easily get an $O(k^d)$-vertex kernel for the d-hitting set problem, which means that the vertex deletion problem to $\mathcal{G}$ also admits an $O(k^d)$-vertex kernel.

2.4.2 The modulator approach

For edge modification problems, the sunflower lemma cannot be used to get a polynomial kernel since we may get a new obstruction by deleting or adding an edge, which means that it is not sufficient only to hit the original obstructions. Here we introduce one of the most common methods, the modulator method, used for the kernelization algorithm of the $\mathcal{G}$ edge modification problems (also used for the vertex deletion problems).

A modulator for a $\mathcal{G}$ edge modification problem is a set of vertices whose removal results in a graph in $\mathcal{G}$. Given an instance (G, k) of a $\mathcal{G}$ edge modification problem, we can get a modulator by packing a maximal set of obstructions if $\mathcal{G}$ is a graph class characterized by finite obstructions, or calling a constant approximation algorithm of the problem if it exists. Suppose M is a modulator of (G, k), then $G - M$ is a graph in $\mathcal{G}$, and then the properties of this graph class can be used to reduce the size of $G - M$. To reduce the size of $G - M$ as much as possible, usually, we can add more vertices in the modulator and make it satisfy some particular property. In [56], the modulator must contain at least one edge of every obstruction, and in [2], the modulator must contain at least nine vertices of every obstruction.

Given an instance (G, k) of a $\mathcal{G}$ edge modification problem, we define that a set $M \subseteq V(G)$ of vertices is a *2-modulator* of a graph G if every obstruction in G intersects M by at least two vertices.

Chapter 3

Cluster Graphs

In this chapter, we introduce a $2k$-vertex kernel for the cluster edge deletion problem. Also, we show that the same algorithm produces a kernel of the same size for the strong triadic closure problem, which, though originally not posed as an edge modification problem, is closely related to cluster edge deletion [101]. As in [28], both algorithms work for the weighted versions of the problems as well.

Cluster graphs is a graph class that is widely used in many applications. In computational biology [146], the genes that manifest similar expression pattern would be identified as a group in the analysis of genes expression data. We can build a cluster graph whose vertices correspond to the genes and there is an edge between two vertices if and only if their corresponding genes manifest similar expression pattern.

3.1 Edge deletion

Our first problem is the cluster edge deletion problem. It is easy to see that the cluster completion problem is trivial: the minimum solution is to add edges to make every component of the input graph complete. Both cluster edge editing and cluster edge deletion are NP-complete and have received wide attentions. The cluster edge deletion problem is NP-hard even on some special graph classes, such as graphs with maximum degree four [100] and $\{2K_2, 3K_1\}$-free graphs [73]. It could be solvable in polynomial time on cographs [73] (where a cograph is a graph that does not contain a P_4 as an induced subgraph) and in time $O(1.42^k + m)$ on general graphs [14].

3.1.1 Previous results

Gruttemeier and Komusiewicz [81] showed a $4k$-vertex kernel for the cluster edge deletion problem. As we have mentioned before, Cao and Chen [28] devised a $2k$-vertex kernel for the cluster edge editing problem and their algorithm actually implies a $2k$-vertex kernel for the cluster edge deletion problem.

3.1.2 The reduction rule

We only have one reduction rule for our kernelization algorithm, which is stated as below.

Rule 3.1.1. *If there is a simplicial vertex v such that $d(N[v]) \le d(v)$, then remove $N[v]$ and decrease k by $d(N[v])$.*

Now, we show the safeness of Rule 3.1.1.

Lemma 3.1.1. *The Rule 3.1.1 is safe.*

Proof. We show that $opt(G) = opt(G - N[v]) + d(N[v])$. Let E_- be an optimal solution to the graph G. We have nothing to show if $N[v]$ makes a separate component of $G - E_-$. In the rest of the proof, $N[v]$ is not a component of $G - E_-$. Let $G[X]$ denote the component of $G - E_-$ that contains v. Since X is a clique and $N[v] \neq X$, we have $X \subseteq N[v]$. In other words, neither X nor $N[v] \setminus X$ is empty. Since any induced subgraph of $G - E_-$ is a cluster graph, the subset of edges in E_- with both endpoints in $V(G) \setminus N[v]$ is a solution to $G - N[v]$. Noting that this solution is disjoint from $E(X, V(G) \setminus X)$, we have

$$\begin{aligned} opt(G) &\geq |E_- \cap E(G - N[v])| + d(X) \\ &\geq opt(G - N[v]) + |X| \cdot |N[v] \setminus X| \\ &\geq opt(G - N[v]) + |X| + |N[v] \setminus X| - 1 \\ &= opt(G - N[v]) + d(v), \end{aligned} \tag{3.1}$$

where the third inequality holds because both $|X|$ and $|N[v] \setminus X|$ are positive integers. For any solution E'_- of $G - N[v]$, the set $E'_- \cup E(N[v], V(G) \setminus N[v])$ is a solution of G. Thus, we have

$$\mathrm{opt}(G) \leq \mathrm{opt}(G - N[v]) + d(N[v]) \leq \mathrm{opt}(G - N[v]) + d(v). \tag{3.2}$$

Therefore, all the inequalities in (3.1) and (3.2) are tight. In other words, if we remove all the edges between $N[v]$ and $V(G) \setminus N[v]$, and then delete an optimal solution to $G - N[v]$, then we have an optimal solution to the graph G. □

Let us mention that the condition of Rule 3.1.1 can be weakened to $d(N[v]) <$

$2d(v) - 1$. We do not prove the stronger statement because it does not improve the analysis of the kernel size, but let us briefly explain why it is true. The bound $\mathrm{opt}(G) \geq \mathrm{opt}(G - N[v]) + 2d(v) - 1$ holds unless $|X| = 1$ or $|N[v] \setminus X| = 1$; see the third inequality of (3.1). In the first case, v itself makes a trivial component, and all the vertices in $N(v)$ are in the same component; this can only happen when there exists another vertex u with $N(v) \subseteq N(u)$. In the second case, a vertex $u \in N(v)$ is incident to all the edges between $N(v)$ and $V(G) \setminus N[v]$. If $d(N[v]) < 2d(v) - 1$, then $\mathrm{opt}(G) \geq \mathrm{opt}(G - N[v]) + 2d(v) - 1$ holds in both cases.

3.2 An improved kernel

With Rule 3.1.1, we can prove Theorem 1.4.1, which we recall here.

Theorem 3.2.1. *The cluster deletion problem admits a $2k$-vertex kernel.*

Proof. Let G be a graph to which Rule 3.1.1 is not applicable. We show that if (G, k) is a yes-instance, then $|V(G)| \leq 2k$. Let E_- be an optimal solution to G, and let $\{v_1, v_2, \ldots, v_r\}$ be the vertices that are not incident to any edge in E_-; they have to be simplicial. For $i = 1, \ldots, r$, the set $N[v_i]$ forms a component of $G - E_-$. Note that for distinct $i, j \in \{1, \ldots, r\}$, the sets $N[v_i]$ and $N[v_j]$ are either the same (when v_i and v_j are true twins) or mutually disjoint: if $N[v_i] \neq N[v_j]$ and there exists $x \in N[v_i] \cap N[v_j]$, then one of xv_i and xv_j needs to be in E_-. We divide the cost of each edge $uv \in E_-$ and assign them to u and v equally. For $i = 1, \ldots, r$, the total cost attributed to all the vertices in $N[v_i]$ is $d(N[v_i])/2$, because Rule 3.1.1 does not apply to v_i. Each of the vertices not in $\bigcup_{i=1}^{r} N[v_i]$ is an endpoint of at least one edge in E_- and therefore bears cost at least $1/2$. Summing them up, we get a lower bound for the total cost:

$$\begin{aligned}|E_-| &\geq \frac{1}{2}\sum_{i=1}^{r} d(N[v_i]) + \frac{1}{2}|V(G) \setminus \bigcup_{i=1}^{r} N[v_i]| \\ &\geq \frac{1}{2}\sum_{i=1}^{r} |N[v_i]| + \frac{1}{2}|V(G) \setminus \bigcup_{i=1}^{r} N[v_i]| \\ &\geq \frac{1}{2}|V(G)|.\end{aligned}$$

The second inequality holds since Rule 3.1.1 does not apply to v_i for $1 \leq i \leq r$. Thus, $|V(G)|/2 \leq |E_-| \leq k$ for a yes-instance, and we can return a trivial no-instance if $|V(G)| > 2k$. This concludes the proof. □

We provide the algorithm in Figure 3.1, whose running time is $O(n^4)$. Step 1 can be done in O(1) time. In Step 2, we can find a simplical vertex v such that $d(N[v]) < d(v)$ in $O(n^3)$ time and do Rule 3.1.1 in $O(n)$ time. Since we delete at least one vertex when we apply Rule 3.1.1, we apply Rule 3.1.1 at most n times. Therefore, Step 2 would take $O(n^4)$ time. Steps 3 and 4 can be completed in $O(n)$ time. Hence, we conclude that the running time of the kernelization algorithm for the cluster edge deletion problem is $O(n^4)$.

```
procedure reduce($G, k$)

1. **if** $k < 0$ **then return** a trivial no-instance;
2. **if** there is a simplicial vertex $u$ such that $d(N[v]) \leq d(v)$ **then**
       apply Rule 3.1.1 and **return** reduce($G, k$);
3. **if** $|V(G)| \leq 2k$ **then return** $(G, k)$;
4. **else return** a trivial no-instance.
```

Figure 3.1: The kernelization algorithm for Cluster Edge Deletion.

3.3 Strong triadic closure

In the original definition, which was motivated by applications in social networks, the *strong triadic closure* problem asks for a partition of the edge set of the input graph into strong edges and weak ones, such that for every two vertices that are linked to a common neighbor with strong edges are adjacent. The objective is to maximize the number of strong edges. For our purpose, it is more convenient to define the problem as follows.

Strong triadic closure

Input: A graph G and a nonnegative integer k.

Output: Is there a set E_- of at most k edges such that the missing edge of every P_3 of $G - E_-$ is in $E(G)$?

Thus, we call the set of weak edges as the solution to the strong triadic closure problem. For any set $E_- \subseteq E(G)$, if $G - E_-$ is a cluster graph, then E_- is also a solution to the strong triadic closure problem: setting all edges in E_- weak, and all other edges strong is a feasible partition of $E(G)$. As illustrated in Figure 3.2, however, a strong triadic closure of a graph can have fewer weak edges than an optimal solution to the cluster edge deletion problem on the same graph.

Surprisingly, Rule 3.1.1 works for the strong triadic closure problem without change.

Lemma 3.3.1. *Rule 3.1.1 is safe for the strong triadic closure problem.*

Proof. We show that $\text{opt}(G) = \text{opt}(G - N[v]) + d(N[v])$. Let E_- be an optimal solution to the graph G. We have nothing to show if $N[v]$ makes a separate component of $G - E_-$. In the rest of the proof, $N[v]$ is not a component of

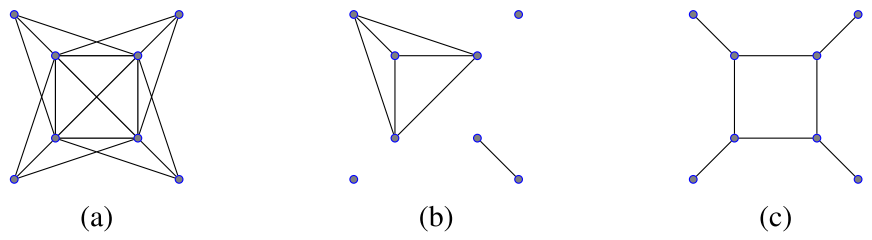


Figure 3.2: The example given by Konstantinidis et al. [101]: (a) the input graph; (b) a maximum cluster subgraph with seven edges; and (c) a maximum strong triadic closure with eight edges.

$G - E_-$. Let X denote the set of vertices with $N[X] = N[v]$, and $Y \subseteq N[v]$ the ends of these edges in $E(N[v], V(G) \setminus N[v]) \setminus E_-$ (i.e., edges between $N[v]$ and $V(G) \setminus N[v]$ that are not in E_-). Note that $X \neq \emptyset$ because $v \in X$, and $Y \neq \emptyset$ because $E(N[v], V(G) \setminus N[v]) \not\subseteq E_-$ (otherwise $N[v]$ is a component of $G - E_-$ by the minimality of E_-).

By definition, the subset of edges in E_- with both ends in $G - N[v]$ is a solution to $G - N[v]$. By the selection of X and Y, every vertex in $N[v] \setminus (X \cup Y)$ is incident to at least one edge in $E_- \cap E(N[v], V(G) \setminus N[v])$. For every $x \in X$ and every $y \in Y$, there exists $z \in V(G) \setminus N[v]$ that is adjacent to y but not x; hence, xyz is a P_3. As a result, all the edges between X and Y have to be in E_-. Thus,

$$\begin{aligned} \text{opt}(G) =& |E_- \cap E(G - N[v])| + |E_- \cap E(N[v], V(G) \setminus N[v])| + |E_- \cap E(N[v])| \\ \geq& \text{opt}(G - N[v]) + |N[v] \setminus (X \cup Y)| + |X| \cdot |Y| \\ \geq& \text{opt}(G - N[v]) + |N[v]| - |X| - |Y| + |X| + |Y| - 1 \\ \geq& \text{opt}(G - N[v]) + |N[v]| - 1 \\ =& \text{opt}(G - N[v]) + d(v), \end{aligned} \tag{3.3}$$

where $|X| \cdot |Y| \geq |X| + |Y| - 1$ because both $|X|$ and $|Y|$ are positive integers. For any solution E'_- of $G - N[v]$, the set $E'_- \cup E(N[v], V(G) \setminus N[v])$ is a solution of G. Thus,

$$\mathrm{opt}(G) \le \mathrm{opt}(G - N[v]) + d(N[v]) \le \mathrm{opt}(G - N[v]) + d(v). \tag{3.4}$$

Therefore, all the inequalities in (3.3) and (3.4) are tight. In other words, if we remove all the edges between $N[v]$ and $V(G) \setminus N[v]$, and then delete an optimal solution to $G - N[v]$, then we have an optimal solution to the graph G. □

The proof of the following theorem is exactly same as that for Theorem 1.4.1, hence omitted.

Theorem 3.3.2. *There is a $2k$-vertex kernel for the strong triadic closure problem.*

For the strong triadic closure problem, we may alternatively state Rule 3.1.1 as follows.

Rule 3.3.1. *If there is a simplicial vertex v such that $d(N[v]) \le d(v)$, then set all the edges in $G[N[v]]$ strong, set all the edges between $N[v]$ and $V(G) \setminus N[v]$ weak, and delete $N[v]$.*

We should remark that our kernelization algorithms for the cluster edge deletion problem and the strong triadic closure problem work for the weighted versions as well; see [28]. By using the cutting lemma, we could modify the Rule 3.1.1 to that if there is a simplicial vertex v such that $\sum_{u \in N[v], w \in N(N[v])} wt(u, w) \le d(v)$, then remove $N[v]$ and decrease k by $\sum_{u \in N[v], w \in N(N[v])} wt(u, w)$.

Chapter 4

Paw-free Graphs

In this chapter, we study the kernelization algorithm for the paw-free completion problem. By Proposition 2.2.2, we get that each component of a paw-free graph is either triangle-free or a complete multipartite graph with at least three parts, which motivates us to use the modulator approach. Note that the deletion of all the vertices in the modulator leaves the graph paw-free. Given an instance (G, k) of the paw-free completion problem, we first construct a 2-modulator M such that any isolated vertex v in $G - M$ cannot dominate all the edges of G_v where G_v is the component of G containing v, and then study the interaction between M and the components of $G - M$, which are triangle-free or complete multipartite.

Independent of our work, Eiben et al. [58] obtain similar results for edge modification problems to paw-free graphs. They also develop a polynomial kernel for the editing problem. They also take the modulator approach and construct a modulator M by finding a maximal packing of edge-disjoint paws. They first apply several simple rules to bound the number of vertices in complete multipartite components in $G - M$ by $O(k^3)$. The gist is to bound the size of triangle-free components of $G - M$. They proved that for any component G' of G, the number of vertices in the triangle-free components of $G' - M$ is at most $4k + 6$. It is easy to bound by k the number of components containing a paw of G, implying that the number of vertices in the triangle-free components of $G - M$ is $O(k^2)$. In our method, we construct a specific modulator and prove that for a component G' of G, the order of $G' - M$ is $c \cdot |G' \cap M|$ where c is a constant, hence we get an $O(k)$ kernel.

4.1 Observations

Given an instance (G, k) of the paw-free completion problem, let M be a 2-modulator of G. Before going details of our kernelization algorithm, we give several simple observations we get for this problem. A simple fact is on the adjacency between a vertex and a (maximal) clique in a paw-free graph.

Proposition 4.1.1. *Let K be a clique in a paw-free graph. If a vertex v is adjacent to K, then $|K \setminus N[v]| \leq 1$.*

Proof. It is trivially true when $|K| \leq 2$. For $|K| \geq 3$, any two vertices in $K \setminus N[v]$, a vertex in $K \cap N(v)$, and v itself induce a paw if $|K \setminus N[v]| > 1$. □

Note that GM is paw-free. The following three propositions characterize the interaction between the modulator M and the components of GM.

Proposition 4.1.2. *If a vertex $v \in M$ forms a triangle with two vertices in a component C of $G - M$, then $N(v) \subseteq M \cup C$.*

Proof. Suppose that uvw is a triangle with $u, w \in V(C)$. Suppose for contrary that there is a vertex x in $N(v)$ that is not in $M \cup C$, which means that x is in another component of $G - M$. Then $\{u, v, w, x\}$ induces a paw with only one vertex in M (see Figure 4.1), contradicting the definition of the 2-modulator. □

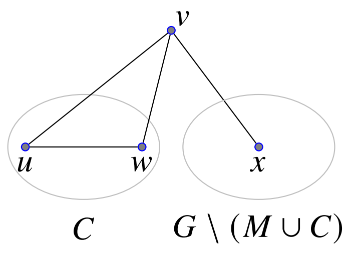


Figure 4.1: Illustration for Proposition 4.1.2.

In other words, if a vertex v in M forms a triangle with a component of $G - M$, then we can see v as a "private" neighbor of this component. As we will see, these components (forming triangles with a single vertex from M) are the focus of our algorithms.

Proposition 4.1.3. *If a vertex $v \in M$ forms a triangle with an edge in a triangle-free component C of $G - M$, then (i) v is adjacent to all the vertices of C; and (ii) C is complete bipartite.*

Proof. Let uvw be the triangle with $u, w \in V(C)$. Suppose that there is a vertex in C nonadjacent to v. We can find a path P in C from this vertex to u, and let x be the last vertex on this path nonadjacent to v. If x is the neighbor of u on this path, we extend the path by appending w after u; otherwise, every vertex on the subpath of P that from u' to u is adjacent to v, where u' is the neighbor of x on this subpath, and the length of this subpath is at least two. Therefore, we always have three consecutive vertices x, u', w', of which only x is nonadjacent to v. Since C is triangle-free, $xw' \notin E(G)$. But then $\{x, u', v, w'\}$ induces a paw with only one vertex in M (see Figure 4.2), a contradiction to the definition of the 2-modulator. Therefore, every vertex in C is adjacent to v.

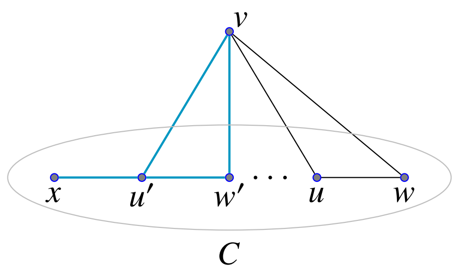


Figure 4.2: Illustration for Proposition 4.1.3.

For the second assertion, suppose that there are three vertices x, y, z in C with only one edge among them, assume it is xy. By the first assertion, we know that v is adjacent to x, y and z. Then they induce a paw with v, where v is the only vertex of the paw that is in M, again contradicting the definition of the 2-modulator. Then C is a triangle-free and $\overline{P_3}$-free component. Thus, C is complete bipartite. $\square$

We say that a triangle-free component of $G - M$ is of type I if there exist two vertices in it that form a triangle with a vertex in M, or type II otherwise. By Proposition 4.1.3, for each type-I triangle-free component, all its vertices have a common neighbor in M. Note that all trivial components of $G - M$ are type-II triangle-free components.

Proposition 4.1.4. *For any complete multipartite component C of $G - M$ and vertex $v \in M$ adjacent to C, the set of vertices in C that are nonadjacent to v is either empty or precisely one part of C.*

Proof. Suppose that the parts of C are $U_1, \ldots, U_p$. We have nothing to prove if all the vertices in C are adjacent to v. In the following we assume that, without loss of generality, v is adjacent to a vertex $u \in U_1$ and nonadjacent to a vertex $w \in U_p$.

We need to argue that v is adjacent to all vertices in the first $p-1$ parts but none in the last part. Any vertex $x \in U_i$ with $1 < i < p$ makes a clique with u and w. It is adjacent to v by the definition of the 2-modulator ($\{u, v, w, x\}$ cannot induce a paw) and Proposition 4.1.1 (see Figure 4.3(a)). Now that v is adjacent to some vertex from another part ($p \geq 3$), the same argument implies $U_1 \subseteq N(v)$. To see $U_p \cap N(v) = \emptyset$, note that a vertex $w' \in U_p \cap N(v)$ would form a paw together with u, v, w (see Figure 4.3(b)), contradicting the definition of the 2-modulator. □

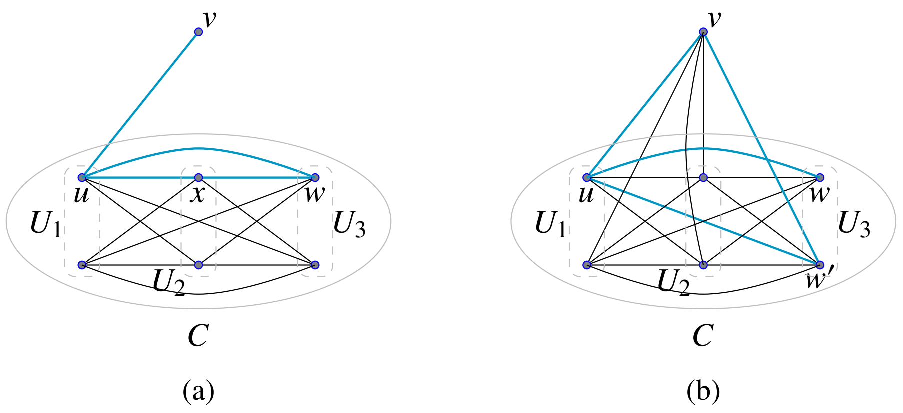


Figure 4.3: Illustration for Proposition 4.1.4.

A *false twin class* of a graph G is a vertex set in which every vertex has the same open neighborhood. It is necessarily independent. The following is immediate from Proposition 4.1.4.

Corollary 4.1.5. *Let C be a complete multipartite component of $G - M$. Each part of C is a false twin class of G.*

Proof. Since C is a complete multipartite component, each part of C is a false twin class of C. By Proposition 4.1.4, we get that the vertices in every part of C have the same neighbors in M. Therefore, each part of C is a false twin class of G. □

We say that a paw-free graph $\widehat{G}$ is a *paw-free completion* of G if $V(G) = V(\widehat{G})$ and $E(G) \subseteq E(\widehat{G})$. The preservation of false twins by all minimum paw-free completions could be of interest to our kernelization algorithm.

Lemma 4.1.6. *Let G be a graph and let E_+ be a minimum set of edges such that $G + E_+$ is paw-free. Any false twin class of G remains a false twin class of $G + E_+$.*

Proof. Let I be a false twin class of G. If I consists of the isolated vertices, then they remain isolated in $G + E_+$ because E_+ is minimum. Hence it remains false twin class. In the rest we may assume that G is connected; otherwise we consider the component of G that contains I. Moreover, if G is paw-free, then E_+ is empty and the claim holds vacuously. Now that G is connected and contains a paw, by Proposition 2.2.2, $G + E_+$ is complete multipartite. It suffices to show that vertices in I are in the same part of $G + E_+$. Suppose for contradiction that $u, v \in I$ are in different parts of $G + E_+$; let P_u and P_v be the parts of $G + E_+$ that contain u and v respectively, and assume without loss of generality that $|P_u| \geq |P_v|$.

Observe that every vertex in $P_u \setminus \{u\}$ is nonadjacent to u in $G + E_+$, hence nonadjacent to u in G, which further implies that it is nonadjacent to v in G because u and v are false twins. Then $G + (E_+ \setminus E_1) \cup E_2$, where $E_1 = \{vx \mid x \in P_u\}$ and $E_2 = \{vx \mid x \in P_v \setminus \{v\}\}$, is also paw-free: Parts P_u and P_v are replaced by $P_u \cup \{v\}$ and $P_v \setminus \{v\}$, while the others remain unchanged. Since $|E_1| = |P_u| > |P_v| - 1 = |E_2|$, this contradicts that E_+ is minimum. This concludes the proof. □

Since all of our reduction rules in this section are very simple and obviously doable in polynomial time, we omit the analysis of their running time.

4.2 Paw-free completion

For the kernelization algorithm for the paw-free completion problem, it is easy to see that the components of G that are paw-free are the easy part to deal with, we could just delete them. Hence, we can do the following reduction rule safely.

Rule 4.2.1. *If a component of G contains no paw, delete it.*

Behind our kernelization algorithm for the paw-free completion problem is the following simple and crucial observation. After Rule 4.2.1 is applied, each remaining component of G contains a paw, hence a triangle, and by Proposition 2.2.2, we need to make it complete multipartite. We say that a vertex v is *major* if v dominates all the edges in the component containing v; note that we always mean the component of the input graph. Recall that a vertex v and an edge xy dominate each other if at least one of x and y is adjacent to v. In a complete multipartite graph, every vertex is major and every edge dominates all its vertices, and hence in a yes-instance, every edge "almost" dominates all the vertices in the component.

Proposition 4.2.1. *Let G be a connected graph containing a paw. For any edge uv and a vertex w that does not dominate uv in G, we need to add at least one of uw and vw to G to make it paw-free.*

Proof. Let E_+ be a set of edges such that $G+E_+$ is paw-free. By Proposition 2.2.2, $G+E_+$ is a complete multipartite graph. Since $uv \in E(G)$, vertices u and v belong to different parts of $G + E_+$. Thus, at least one of them is not in the same part as w, and the edge between it and w is in E+. $\square$

In other words, for each edge uv, the number of edges we need to add to u and v is at least the number of vertices uv does not dominate. As hinted by Proposition 4.2.1, the way we bound the kernel size for the paw-free completion problem is to show that, after applying some reductions, the minimum number of edges we need to add to G to make it paw-free is linear in $|V(G)|$. We need to tweak the modulator for this problem: We also take the degree-one vertices of all paws in G, and the purpose is to simplify the disposal of triangle-free components of $G - M$. As a result of Proposition 4.2.1, if there are more than $2k$ non-major vertices, then we know (G, k) is a no-instance. Hence, we focus on major vertices, for which we consider the components G' of G one by one. If too many isolated vertices in $G' - M$ are major, then we can remove most of them (Rule 4.2.2). No vertex in $G' - M$ can be major if there are two or more nontrivial components in $G' - M$. In the rest, there is precisely one type-ı triangle-free component or precisely one complete multipartite component, but not both, in $G' - M$. In this case, no isolated vertex in $G' - M$ can be major (Proposition 4.2.4), and hence we are concerned with the only nontrivial component of $G' - M$. For a type-ı triangle-free component C of $G' - M$, we show that if C satisfies any of the six simple conditions stipulated in Lemma 4.2.8, then we need to add at least $|V(C)|/32$ edges to G'. Otherwise, we can apply Rule 4.2.3 to reduce G'. The disposal of a complete multipartite component is very similar (Lemma 4.2.10 and Rule 4.2.4). We summarize the algorithm in Figure 4.7 and use it to prove Theorem 1.4.3. Formally, the 2-modulator M for the completion problem is constructed as in Figure 4.4.

It is easy to see that the modulator constructed in Figure 4.4 is a 2-modulator. Now we show some properties of a 2-modulator.

Lemma 4.2.2. *For each component G' of G, we need to add at least $|M \cap V(G')|/4$ edges to G' to make it paw-free.*

```
1.    $\mathcal{F} \leftarrow \emptyset$; $M \leftarrow \emptyset$;
2.    **for each** paw $F$ of $G$ **do**
2.1.      **if** $|F \cap F'| \leq 1$ for each paw $F'$ in $\mathcal{F}$ **then**
              $\mathcal{F} \leftarrow \mathcal{F} \cup \{F\}$;
              add the vertices of $F$ to $M$;
2.2.      **else** add the degree-one vertex of $F$ to $M$;
3.    **return** $M$.
```

Figure 4.4: The construction of the 2-modulator for G.

Proof. Let A be the set of vertices of those paws in G' we greedily added to M, and B the set of degree-one vertices of all other paws in G'; thus, $A \cup B = M \cap V(G')$. For each paw F added to A, at least one of its missing edges needs to be added to G' to make it paw-free. This edge is not in any previous selected paw, as otherwise we should not have added F to A. Therefore, we need to add at least $|A|/4$ edges to $G[A]$ to make it paw-free. On the other hand, each vertex v in B is the degree-one vertex of some paw F, (it is possible that all other three vertices of F are in A) and therefore we need to add at least one edge incident to v. Therefore, we need to add at least $|B|/2$ edges incident to vertices in B to G' to make it paw-free. Note that these two sets of edges we need to add are disjoint (since the paws we forbid by adding edges are different). The total number of edges we need to add to G' to make it paw-free is at least

$$\frac{|A|}{4} + \frac{|B|}{2} \geq \frac{|A \cup B|}{4} = \frac{|M|}{4}.$$

This concludes the proof. □

Corollary 4.2.3. *If (G, k) is a yes-instance, then M contains at most $4k$ vertices.*

Proof. Let $C_1, C_2, \ldots, C_\ell$ be the components of G and M the 2-modulator constructed in Figure 2. Let E_+ be a solution to (G, k), and let $E_i \subseteq E_+$ be the edges that need to be added to C_i to make it paw-free. By Lemma 4.2.2, we get that $|E_i| \geq |M \cap V(C_i)|/4$ and then $|E_+| = \sum_{i=1}^{\ell} |E_i| \geq \sum_{i=1}^{\ell} |M \cap V(C_i)|/4 = |M|/4$. Since $|E_+| \leq k$, therefore $|M| \leq 4k$. □

We proceed only when $|M| \leq 4k$. Recall that a vertex v is major if v dominates all the edges in the component containing v. By Proposition 4.2.1, for each non-major vertex v, we need to add at least one edge incident to v to make the graph paw-free. Therefore, after the application of Rule 4.2.1, the total number of non-major vertices in a yes-instance is at most $2k$. It remains to deal with major vertices in $V(G) \setminus M$, for which we consider the components G' of G one by one; let $M' = M \cap V(G')$.

Proposition 4.2.4. *The following hold for major vertices of G'.*

(i) No vertex in a nontrivial type-II triangle-free component of $G' - M'$ can be major.

(ii) If two components in $G' - M'$ are nontrivial, then no vertex in $G' - M'$ is major.

(iii) If there is an isolated vertex in $G' - M'$ that is major, then all vertices in $G' - M'$ are isolated.

Proof. Let C be a nontrivial type-II triangle-free component of $G' - M'$. Since G' contains a paw, there must be a triangle in M' (by the construction of M); let it be uvw. If a vertex x in C is major, then x is adjacent to at least two of u, v, and w; assume without loss of generality, $u, v \in N(x)$. Since C is a nontrivial component, we can find a neighbor y of x. Since C is a type-II triangle-free component, the vertex y is adjacent to neither of u and v. Then y is a degree-one vertex of the paw induced by $\{u, v, x, y\}$, and should be in M'. This contradicts to the construction of M, and thus (i) holds.

Let C_1 and C_2 be two nontrivial components of $G' - M'$. Then there is an edge in each of C_1 and C_2. If there is a major vertex v in $G' - M'$, then v must be adjacent to both C_1 and C_2, which is impossible. This concludes assertion (ii).

Suppose that an isolated vertex v in $G' - M'$ is major. By definition, every edge in G' has one end adjacent to v, and this end must be in M', which means that there is no edge in $G' - M'$. This concludes assertion (iii). $\square$

By Proposition 4.2.4(ii), we may assume that there is at most one nontrivial component in $G' - M'$. The next two propositions are on independent sets I of G'. The first is about the cost of separating vertices in I into more than one part; it also means that a sufficiently large independent set cannot be separated.

Proposition 4.2.5. *Let G' be a connected graph containing a paw, and let I be an independent set of G'. If we do not add all the missing edges between I and $N(I)$, then we need to add at least $|I| - 1$ edges among I to G' to make it paw-free.*

Proof. Let E_+ be a set of edges such that $G' + E_+$ is paw-free. Since G' contains a paw, $G' + E_+$ is a complete multipartite graph by Proposition 2.2.2. If all vertices in I are in the same part of $G' + E_+$, then all the missing edges between I and $N(I)$ are in E_+ and we are done. Otherwise, vertices in I are in at least two parts of $G' + E_+$ (see Figure 4.5), and the number of edges among them in E_+ is at least $|I| - 1$ (when one part has one vertex and the rest are in another part). $\square$

Let I be an independent set of G' such that every vertex in I is adjacent to all vertices not in I. If $V(G') \setminus I$ is also an independent set, then G' is complete

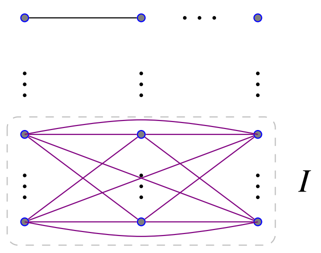


Figure 4.5: Illustration for Lemma 4.2.5. The graph $G' + E_+$ is a complete multipartite graph, all violet edges should be in E_+ when vertices in I are in more than one part of $G' + E_+$.

bipartite and paw-free. Hence, we assume that G' contains a triangle, and we need to make it complete multipartite. Clearly, the set I has to form a separate part by itself. If $V(G') \setminus I$ is connected and contains a triangle, then any solution to $G' - I$ is a solution to G', and thus we can remove I. Otherwise, we need to keep one vertex in I to make sure the component contains a triangle. The second assertion of the following proposition is not used in this thesis.

Proposition 4.2.6. *Let I be an independent set in a component G' of a graph G. If every vertex in I is adjacent to every vertex in $V(G') \setminus I$, then (G, k) is a yes-instance if and only if $(G - (I \setminus \{v\}), k)$ is a yes-instance for any $v \in I$. Moreover, if $G - I$ is connected, then (G, k) is a yes-instance if and only if $(G - I, k)$ is a yes-instance.*

Proof. It suffices to show the if direction. Suppose that $(G - (I \setminus \{v\}), k)$ is a yes-instance for any $v \in I$, then we show that (G, k) is a yes-instance. It is vacuously true when G is paw-free, and henceforth we assume otherwise. Suppose that E_+ is a solution to $(G - I', k)$, where $I' = I \setminus \{v\}$. Note that $G' - I'$ is connected because every other vertex is adjacent to v. By Proposition 2.2.2, $(G' - I') + E_+$ is a complete multipartite graph; let $U_1, \ldots, U_\ell$ be its parts. Since v is adjacent to every other vertex, it is in a size-one part; without loss of generality, let it be U_ℓ.

On the other hand, I remains an independent set and adjacent to every vertex in $V(G') \setminus I$ in $G + E_+$. Therefore, $G' + E_+$ is a complete multipartite graph with parts $U_1, \ldots, U_{\ell-1}, I$, hence paw-free. The proof of the second assertion is very similar and omitted. □

We are now ready for our reduction rules. We start from type-ɪɪ triangle-free components of $G' - M'$. If there is a nontrivial type-ɪɪ triangle-free component in $G' - M'$, then by Proposition 4.2.4(i, iii), no vertex in $G' - M'$ can be major. Now suppose that all vertices in $G' - M'$ are isolated and some of them are major. We observe that if too many of them are major, then any optimal solution puts the major ones of them and their non-neighbors in the same part. Thus, by Proposition 4.2.6, we can remove all but one of these vertices, after adding certain edges.

Rule 4.2.2. *Let I be the set of isolated vertices in $G' - M'$ that are major. If $|I| \geq |M'|$, then add all the missing edges between $V(G') \setminus N(I)$ and $N(I)$; decrease k accordingly; and remove all but one vertex in $V(G') \setminus N(I)$ from G'.*

Lemma 4.2.7. *Rule 4.2.2 is safe.*

Proof. We argue first that I is a false twin class of G; i.e., $N(v_1) = N(v_2)$ for any pair of vertices $v_1, v_2 \in I$. Note that v_1 and v_2 are not adjacent. For any vertex $x \in N(v_1)$, we must have $x \in N(v_2)$ as well because v_2 dominates the edge xv_1. Therefore, $N(v_1) \subseteq N(v_2)$. For the same reason, $N(v_2) \subseteq N(v_1)$, and then $N(v_1) = N(v_2)$.

Since each vertex in I is major, $V(G') \setminus N[I]$ is an independent set. Then $V(G') \setminus N(I)$, which is $\left(V(G') \setminus N[I]\right) \cup I$, is also an independent set. We argue that any optimal solution E_+ of G' contains all the missing edges between $V(G') \setminus N(I)$

and $N(I)$. By Lemma 4.1.6, the set I remains an independent set in $G' + E_+$, which is complete multipartite. Suppose that some vertices in $V(G') \setminus N(I)$ are not in the same part as I; let X be the set of all such vertices. We consider another solution E'_+ obtained from E_+ as follows: removing all edges incident to X and all edges incident to I, and then adding all missing edges between X and $N(I)$. Since $|I| \geq |M'| \geq |N(I)| > |N(I) \setminus N(x)|$ for each $x \in X$ (the last one holds because G' is connected and vertices in I are major), $|E'_+| < |E+|$. This contradicts that E_+ is optimal.

After adding all the missing edges between $V(G') \setminus N(I)$ and $N(I)$, every vertex in $V(G') \setminus N(I)$ is adjacent to every vertex in $N(I)$, and the correctness of removing all but one vertex in $V(G') \setminus N(I)$ from G follows from Proposition 4.2.6. □

Henceforth, the subgraph $G' - M'$ has precisely one type-ɪ triangle-free component or one complete multipartite component, but not both (by Proposition 4.2.4). Each part of such a component is an independent set (recall that a type-ɪ triangle-free component is complete bipartite by Proposition 4.1.4). We try to relate the order of this component to $|M'|$ or the number of major vertices in G'.

Lemma 4.2.8. *Let $G[U]$ be a type-ɪ triangle-free component of $G' - M'$ and let $L \uplus R$ be the bipartition of $G[U]$ with $|L| \geq |R|$. If any of the following conditions is satisfied, then we need to add at least $|U|/32$ edges to G' to make it paw-free.*

(i) $|L| \leq 4|M'|$;

(ii) *there is an edge in $G' - N[L]$;*

(iii) $V(G') \neq N[U]$ *and* $|L| \leq 2|R|$;

(iv) *there are $|L|/2$ or more missing edges between L and $N(L)$;*

(v) $|L| \leq |R| + |M'|$ *and* $G' - N[R]$ *has an edge; or*

(vi) $|L| \leq |R| + |M'|$ *and there are* $|R|/2$ *or more missing edges between* R *and* $N(R)$.

Proof. (i) If $|L| \leq 4|M'|$, then $|U| = |L| + |R| \leq 2|L| \leq 8|M'|$, and it follows from Lemma 4.2.2.

(ii) By Proposition 4.2.1, we need to add at least $|L| \geq |U|/2$ edges.

(iii) Since $G[U]$ is complete bipartite and $|L| \geq |R|$, we can find a matching of size $|R|$ between L and R. By Proposition 4.2.1, for each vertex $v \in V(G') \setminus N[U]$, the number of edges between v and U we need to add is at least $|R| = (2|R| + |R|)/3 \geq (|L| + |R|)/3 = |U|/3$.

(iv) By Proposition 4.2.5, we need to add at least $|L|/2 \geq |U|/4$ edges.

In the rest, (v) and (vi), we have $|L| \leq |R| + |M'|$. We may assume none of the previous conditions is satisfied. Therefore, we can get that $|L| > 4|M'|$, which means $|L| \leq 2|R|$. Also note that the proofs for these two conditions are almost the same as conditions (ii) and (iv) respectively.

(v) By Proposition 4.2.1, we need to add at least $|R| \geq |U|/3$ edges.

(vi) By Proposition 4.2.5, we need to add at least $|R|/2 \geq |U|/6$ edges. □

We say that a type-ı triangle-free component C of $G' - M'$ is reducible if none of the conditions in Lemma 4.2.8 holds true.

Rule 4.2.3. *Let* C *be a type-ı triangle-free component of* $G' - M'$ *and let* $L \uplus R$ *be the bipartition of* C *with* $|L| \geq |R|$. *If* C *is reducible, then add all the missing edges between* L *and* $N(L)$ *and all the missing edges between* $V(G') \setminus N[L]$ *and* $N(L)$*; decrease* k *accordingly; and remove all but one vertex from* $V(G') \setminus N(L)$.

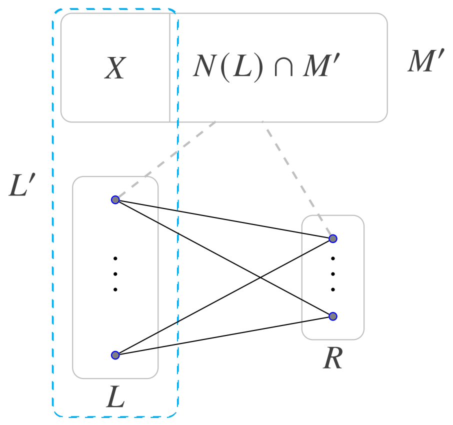


Figure 4.6: Illustration of notations L, R, L', X, M', and $M' \cap N(L)$.

Lemma 4.2.9. *Rule 4.2.3 is safe.*

Proof. Let $X = V(G') \setminus N[L]$ and $L' = L \cup X$ (see in Figure 4.6); note that L' is an independent set because condition (ii) of Lemma 4.2.8 is not true. We show the existence of a solution that contains all the missing edges between L' and $N(L)$. Let E_+ be any solution to (G, k), we may assume that it does not contain edges between G' and other components of G. Let $E' = E_+ \cap \binom{V(G')\setminus L'}{2}$. We show that the set E'_+, consisting of E' together with all the missing edges between L' and $N(L)$, is also a solution of (G, k). By Proposition 2.2.2, $G' + E_+$ is a complete multipartite graph, then its subgraph $(G' - L') + E'$ is complete bipartite or complete multipartite. Therefore, $G' + E'_+$ is a complete multipartite graph, where L' forms a single part. Since E_+ and E'_+ have the same set of edges incident to $V(G') \setminus L'$, to show $|E'_+| \leq |E_+|$ we may focus on their edges incident to L'.

Case 1, $|L| > |R| + |M'|$. If E_+ contains all the missing edges between L and $N(L)$, then E_+ and E'_+ have the same set of edges between L and $N(L)$. Otherwise, by Proposition 4.2.5, E_+ contains at least $|L| - 1 \geq |L|/2$ edges among L; since condition (iv) of Lemma 4.2.8 is not true, this is strictly more than the number of edges in E'_+ between L and $N(L)$. It suffices to show that the number of edges incident to X in E_+ is at least $\sum_{x \in X} |N(L) \setminus N(x)|$, the number of edges incident to X in E'_+. Note that $N(L) \subseteq M' \cup R$, and hence $|N(L)| \leq |R| + |M'| < |L|$. We claim that there is a matching from $N(L)$ to L of size $|N(L)|$; otherwise there exists a vertex set $Y \subseteq N(L)$ with $|N(Y) \cap L| < |Y|$ by Hall's theorem, but then the missing edges between L and $N(L)$ is at least $|Y| \cdot |L \setminus N(Y)| > |Y| \cdot (|L| - |Y|) \geq |L| - 1 \geq |L|/2$, and condition (iv) of Lemma 4.2.8 would be true. For each vertex $x \in X$, we can find from the matching at least $|N(L) \setminus N(x)|$ edges that are not dominated by x. Since these edges do not share any end, by Proposition 4.2.1, at least $|N(L) \setminus N(x)|$ edges in E_+ are incident to x. Thus, the number of edges in E_+ incident to X is at least $\sum_{x \in X} |N(L) \setminus N(x)|$.

Case 2, $|L| \leq |R| + |M'|$. Since C is reducible, none of the conditions in Lemma 4.2.8 is satisfied. In particular, $|L| > 4|M'|$ (condition i), which means $|R| > 3|M'|$ and $|L| \leq 2|R|$. We must have $V(G') = N[C]$ (condition iii). Moreover, by conditions (ii) and (v), both $V(G') \setminus N[L]$ and $V(G') \setminus N[R]$ are independent sets, and by conditions (iv) and (vi), the number of missing edges between L and $N(L)$ and the number of missing edges between R and $N(R)$ are smaller than $|L|/2$ and $|R|/2$, respectively.

If $|P \cap L| < |L|/2$ for every part P in $G' + E_+$,then every vertex v in L' is incident to more than $|L|/2$ edges in E_+. The total number of edges in E_+ that are incident to L' is more than $\frac{1}{2}(|L| \cdot |L|/2) + |X| \cdot |L|/2$. On the other hand, the total number of edges in E'_+ that are incident to L' is at most the sum of the number of missing edges between L and $N(L)$, the number of missing edges between R and $N(R)$, and the number of missing edges between X and $M' \setminus X$, which is at most $|L|/2 + |R|/2 + |X| \cdot |M' \setminus X|$. Thus, we have $|E'_+| \leq |E_+|$.

Now suppose that there is a part P in $G' + E_+$ with $|P \cap L| \geq |L|/2$. If $L' = P$, then $|E'_+| = |E_+|$ and we are done. In the rest $L' \neq P$. Since $|P \cap L| \geq |L|/2$ and there are less than $|L|/2$ missing edges between $N(L)$ and L by condition (iv) of Lemma 4.2.8, no vertex in $N(L)$ can be in P. Therefore $P \subseteq L'$, and $L' \not\subseteq P$. If there is precisely one vertex v in $L' \setminus P$, then all the $|L'| - 1$ edges between v and P are in E_+. But the number of missing edges between v and $M' \cup R$ (in E'_+) is less than

$$|M'| + |R|/2 \leq |L|/4 + |L|/2 < |L| - 1 \leq |L'| - 1.$$

Otherwise, suppose that $|L' \setminus P| = t > 1$, then at least $t(|L'| - t) \geq t|L|/2$ edges among L' are in E_+. But the number of missing edges between $L' \setminus P$ and $M' \cup R$ (in E'_+) is less than $t|M'| + |R|/2 \leq t|L|/4 + |L|/2 \leq t|L|/2$.

We have thus proved that it is safe to add all the missing edges between L' and $N(L)$. After that, every vertex in L' is adjacent to every vertex in $G' - L'$, and the correctness of removing all but one vertex in L' from G follows from Proposition 4.2.6. □

In the last we consider the complete multipartite components of $G' - M'$.

Lemma 4.2.10. *Let $G[U]$ be a complete multipartite component of $G' - M'$, and let P^* be a largest part of $G[U]$. If any of the following conditions is satisfied, then we need to add at least $|U|/12$ edges to G' to make it paw-free.*

(i) $|U| \leq 3|M'|$;

(ii) there is an edge in $G' - N[U]$;

(iii) $|P^*| > 2|U|/3$ *and* $G' - N[P^*]$ *has an edge;*

(iv) $|P^*| \leq 2|U|/3$ *and* $V(G') \neq N[U]$*; or*

(v) $|P^*| \leq 2|U|/3$ *and* $V(G') = N[U]$*, and for every part* P *of* $G[U]$*,*

- $G' - N[P]$ *contains an edge, or*
- *there are at least* $|P|$ *missing edges between* $V(G') \setminus N[P]$ *and* $N(P)$*.*

Proof. (i) If $|U| \leq 3|M'|$, then it follows from Lemma 4.2.2.

(ii) By Proposition 4.2.1, we need to add at least $|U|$ edges.

(iii) By Proposition 4.2.1, we need to add at least $|P^*| > 2|U|/3$ edges.

(iv) Let E_+ be a minimum set of edges such that $G + E_+$ is paw-free. By Corollary 4.1.5 and Lemma 4.1.6, the vertices in any part of $G[U]$ remain in the same part in $G' + E_+$. Since $V(G') \neq N[U]$, there is a vertex $v \in V(G') \setminus N[U]$, and the missing edges between v and all but one part of $G[U]$ are in E_+. Since P^* is a largest part and $|P^*| \leq 2|U|/3$, we need to add at least $|U|/3$ edges.

For (v), we show that we need to add at least $|P|$ edges incident to $P \cup (V(G') \setminus N[P])$. By Proposition 4.1.4 and the fact $V(G') = N[U]$, the sets $V(G') \setminus N[P]$ for different parts are disjoint. As a result, each edge is counted at most twice, and the total number of edges we need to add is $|U|/2$. If there is an edge in $G' - N[P]$, then it follows from Proposition 4.2.1. Now there is no edge in $G' - N[P]$ and there are more than $|P|$ missing edges between $V(G') \setminus N[P]$ and $N(P)$. Let $X = V(G') \setminus N[P]$, and let E_+ be a minimum set of edges such that $G + E_+$ is paw-free. By Corollary 4.1.5 and Lemma 4.1.6, the vertices in any part of $G[U]$ remain in the same part in $G' + E_+$. If all vertices in X are in the same part as P in $G' + E_+$, then all missing edges between X and $N(P)$ are in E_+. Otherwise, there is a vertex x in X that is not in the same part as P in $G' + E_+$, and all edges between x

and P are in E_+. In either case, we need to add at least $|P|$ edges incident to $P \cup X$. This concludes the proof. □

We say that a complete multipartite component of $G' - M'$ is *reducible* if none of the conditions in Lemma 4.2.10 holds true.

Rule 4.2.4. *Let $G[U]$ be a reducible complete multipartite component of $G' - M'$ and P^* a largest part of $G[U]$.*

1. *If $|P^*| > 2|U|/3$, then add all the missing edges between $V(G') \setminus N(P^*)$ and $N(P^*)$; decrease k accordingly; and remove all but one vertex in $V(G') \setminus N(P^*)$ from G'.*

2. *Otherwise, find a part P such that $V(G') \setminus N(P)$ is an independent set and there are less than $|P|$ missing edges between $V(G') \setminus N(P)$ and $N(P)$. Add all the missing edges between $V(G') \setminus N(P)$ and $N(P)$; decrease k accordingly; and remove all but one $V(G') \setminus N(P)$ from G'.*

Lemma 4.2.11. *Rule 4.2.4 is safe.*

Proof. Let E_+ be a minimum set of edges such that $G' + E_+$ is paw-free; note that it does not have edges between G' and other components of G. By Proposition 2.2.2, $G' + E_+$ is a complete multipartite graph. By Corollary 4.1.5 and Lemma 4.1.6, vertices in the same part of $G[U]$ remain in the same part in $G' + E_+$.

Case 1: $|P^*| > 2|U|/3$. Let $X = V(G') \setminus N[P^*]$. Since $G[U]$ is reducible, none of conditions in Lemma 4.2.10 is satisfied. In particular, $|U| > 3|M'|$ (condition i) and X is an independent set (condition iii). For each vertex $x \in X$, if it is not in the same part of $G' + E_+$ as P^*, then all the $|P^*|$ missing edges between x and P^*

are in E_+. Since

$$|N(P^*)| \leq |(U \setminus P^*) \cup M'| = |U \setminus P^*| + |M'| < |U|/3 + |U|/3 < |P^*|,$$

putting x to the same part as P^* minimizes the number of edges incident to it. After adding all the missing edges between X and $N(P^*)$, every vertex in the independent set $P^* \cup X$ is adjacent to every vertex in $V(G') \setminus (P^* \cup X) = N(P^*)$. It is thus safe to remove all but one vertex in $P^* \cup X$ from G' by Proposition 4.2.6.

Case 2: $|P^*| \leq 2|U|/3$. Since the component $G[U]$ is reducible, none of conditions in Lemma 4.2.10 is satisfied. In particular, we have $V(G') = N[U]$ (condition iv) and we can find a part P of $G[U]$ such that $V(G') \setminus N(P)$ is an independent set and the missing edges between $V(G') \setminus N(P)$ and $N(P)$ is less than $|P|$ (condition v). Let $X = V(G') \setminus N[P]$. For each vertex $x \in X$, if it is not in the same part of $G' + E_+$ as P, then all the $|P|$ missing edges between x and P are in E_+. This is more than the total number of missing edges between X and $N(P)$. Hence, putting $X \cup P$ in the same part minimizes the number of edges incident to it. After adding all the missing edges between X and $N(P)$, all vertices in the independent set $P \cup X$ are adjacent to $V(G') \setminus (P \cup X) = N(P)$. It is thus safe to remove all but one vertex in $P \cup X$ from G' by Proposition 4.2.6. □

We summarize our kernelization algorithm for the paw-free completion problem in Figure 4.7.

```
procedure reduce($G, k$)

0.   if $k < 0$ then return a trivial no-instance;
1.   remove all paw-free components from $G$;
2.   construct a 2-modulator $M$;
3.   if $|M| > 4k$ then return a trivial no-instance;
4.   if more than 2k non-major vertices then return a trivial no-instance;
5.   for each component $G'$ of $G$ do
5.1.     $M' \leftarrow V(G') \cap M$;
5.2.     if two components in $G' - M'$ are nontrivials then
             skip $G'$ and goto 5;
5.3.     if there are at least $|M'|$ isolated major vertices in $G' - M'$ then
             apply Rule 4.2.2 and return reduce($G, k$);
5.4.     if $G' - M'$ has a type-ɪ triangle-free component $C$ then
             if $C$ is reducible then apply Rule 4.2.3 and return reduce($G, k$);
5.5.     if $G' - M'$ has a complete multipartite component $C$ then
             if $C$ is reducible then apply Rule 4.2.4 and return reduce($G, k$);
6.   if $|V(G)| \leq 38k$ then return $(G, k)$;
7.   else return a trivial no-instance.
```

Figure 4.7: The kernelization algorithm for Paw-free Completion.

4.3 A linear kernel

We use our kernelization algorithm in Figure 4.7 to prove Theorem 1.4.3, which we recall here.

Theorem 4.3.1. *The paw-free completion problem has a $38k$-vertex kernel.*

Proof. We use the algorithm described in Figure 4.7. The correctness of steps 0 and 1 follows from the definition of the problem and Rule 4.2.1 respectively. Steps 2 and 3 are justified by Lemma 4.2.2. Step 4 is correct because of Proposition 4.2.1, and after that step 5 deals with the components of G separately. The cost of a component G' of G is the minimum number of edges we need to add to G' to make it paw-free. We show that the cost of G' is at least $1/32$ of the number of major vertices in $V(G') \setminus M'$.

If there are two or more nontrivial components in $G' - M'$, then by Proposition 4.2.4(ii) and Proposition 4.2.1, the cost of G' is at least $|M'|/4 + |V(G') \setminus M'|/2 > |V(G')|/4$, and thus G' can be skipped in step 5.2. Let r be the number of isolated vertices in $G' - M'$ that are major. If $r \geq |M'|$, then step 5.3 calls Rule 4.2.2 to remove most of these vertices, and the correctness is given in Lemma 4.2.7. If $0 < r < |M'|$, then by Proposition 4.2.4(iii), every component of $G' - M'$ is trivial, and the cost of G' is at least $|M'|/4 > r/4$. Henceforth, $G' - M'$ has precisely one type-ɪ triangle-free component or one complete multipartite component, while no isolated vertex in $G' - M'$ is major. The algorithm enters step 5.4 if there is a type-ɪ triangle-free component C in $G' - M'$. If C is reducible, the correctness of Rule 4.2.3 is given in Lemma 4.2.9; otherwise, the cost of G' is at least $|C|/32$ by Lemma 4.2.8. The algorithm enters step 5.5 if there is a complete multipartite component C in $G' - M'$. If C is reducible, the correctness of Rule 4.2.4 is given in Lemma 4.2.11; otherwise, the cost of G' is at least $|C|/12$ by Lemma 4.2.10.

When the algorithm reaches step 6, none of Rules 4.2.2, 4.2.3, and 4.2.4 is applicable. There are at most $4k$ vertices in M, at most $2k$ non-major vertices. On the other hand, for each other vertex, there is an amortized cost of at least $1/32$. Therefore, if (G, k) is a yes-instance, then the number of vertices in G is at most $38k$, and this justifies steps 6 and 7.

We now analyze the running time of this algorithm. When each time the algorithm calls itself in step 5.3, 5.4, or 5.5, it removes at least one vertex from the graph. Therefore, the recursive calls can be made at most n times. On the other hand, each step clearly takes polynomial time. Therefore, the algorithm returns in polynomial time. □

We spare little effort in getting a better kernel size and more efficient implementation. For the running time, e.g., we do not need to call the algorithm after each reduction (application of Rules 4.2.3 and 4.2.4). We believe that, with more careful analysis, one may show that our kernel is between $12k$ and $16k$ vertices. However, to find a kernel of less than $10k$ vertices for the problem or to implement it in linear time is quite a challenge.

Chapter 5

Trivially Perfect Graphs

In this chapter, we propose a very simple kernelization algorithm, which has only two simple reduction rules, and the resulting kernel contains at most $2k^2 + 2k$ vertices. The forbidden induced subgraphs of trivially perfect graphs are P_4 and C_4. Note that adding the edge to connect the two ends of a P_4 merely turns it into a C_4. Thus, in each P_4 or C_4, there are two missing edges (call them potential missing edges) such that every solution needs to contain at least one of them. Note that each vertex of the P_4 or C_4 is an end of one of the two missing edges. Our first rule is the most routine for this kind of problems, namely, adding a missing edge if it is one of the two possible missing edges in $k + 1$ or more P_4's and C_4's. Our second rule removes all vertices that are not contained in any P_4 or C_4 of G. Now the analysis is similar as Buss and Goldsmith's kernelization algorithm for the vertex cover problem [21]. Since every solution contains at least one of the pair of potential missing edges (for some P_4 or C_4), and since each potential edge is in at most k pairs, there cannot be more than k^2+k potential edges in a yes-instance. On the other hand, every vertex is in a P_4 or C_4, hence an endpoint of some potential edge. We are thus safe to return a trivial no-instance when the number of vertices of the input graph is larger than $2k^2 + 2k$.

5.1 Properties of trivially perfect graphs

Trivially perfect graphs was first studied by Wolk [162, 161], and was named by Golumbic [77]. Recall that trivially perfect graphs is characteized by forbidden induced subgraphs P_4 and C_4. If there is a pair of adjacent vertices u, v such that neither $N[u] \setminus N[v]$ nor $N[v] \setminus N[u]$ is empty, then they are contained in a P_4 or C_4. An additional important characterization of trivially perfect graphs is provided in [165]. According to this characterization, a trivially perfect graph can be generated by adding a universal vertex and performing a disjoint union with the single-vertex graph K_1. By these characterizations of trivially perfect graphs, we can easily see that the following theorem is correct.

Theorem 5.1.1 ([162, 165]). *The following are equivalent for a graph H.*

(i) H is a trivially perfect graph.

(ii) Every connected induced subgraph of H contains a universal vertex.

(iii) For every pair of adjacent vertices u and v, one of $N[u]$ and $N[v]$ is a subset of the other.

5.2 Trivially perfect completion

We say that a trivially perfect graph $\widehat{G}$ is a *trivially perfect completion* of G if $V(G) = V(\widehat{G})$ and $E(G) \subseteq E(\widehat{G})$, and it is minimal if there is no other trivially perfect completion $\widehat{G}'$ of G with $E(G) \subseteq E(\widehat{G}') \subset E(\widehat{G})$. The following two observations are very simple.

Proposition 5.2.1. *Let H be a connected graph, and let $\widehat{H}$ be a minimal (minimum) trivially perfect completion of H. For any universal vertex u of $\widehat{H}$, the graph $\widehat{H} - u$ is a minimal (minimum) trivially perfect completion of $H - u$.*

Proof. Adding u as a universal vertex to any trivially perfect completion of $H - u$, we end with a trivially perfect completion of H (by Theorem 5.1.1). $\square$

Lemma 5.2.2. *Let $\widehat{G}$ be a minimal trivially perfect completion of a graph G, and let u and v be two vertices of G. If $N_G[u] \subseteq N_G[v]$, then $N_{\widehat{G}}[u] \subseteq N_{\widehat{G}}[v]$.*

Proof. Suppose for contradiction, $N_{\widehat{G}}[u] \nsubseteq N_{\widehat{G}}[v]$. By Theorem 5.1.1(iii), $N_{\widehat{G}}[v] \subset N_{\widehat{G}}[u]$. Since $N_G[u] \subseteq N_G[v] \subseteq N_{\widehat{G}}[v] \subset N_{\widehat{G}}[u]$, it follows that $xu \notin E(G)$ for every $x \in N_{\widehat{G}}[u] \setminus N_{\widehat{G}}[v]$. Let $E_+ = E(\widehat{G}) \setminus E(G)$. We consider

$$E'_+ = E_+ \setminus \{xu \mid x \in N_{\widehat{G}}[u] \setminus N_{\widehat{G}}[v]\} \text{ and } \widehat{G}' = G + E'_+.$$

Then $N_{\widehat{G}'}[u] = N_{\widehat{G}'}[v]$. Since $\widehat{G}' - u = \widehat{G} - u$, it is a trivially perfect graph. On the other hand, since u and v are true twins of $\widehat{G}'$, the graph $\widehat{G}'$ is also a trivially perfect graph. But since $E(G) \subseteq E(\widehat{G}') \subset E(\widehat{G})$, we have a contradiction to the minimality of $\widehat{G}$. $\square$

If a vertex v is not contained in any P_4 or C_4, then for every neighbor u of v, one of $N[u]$ and $N[v]$ is a subset of the other. Let (G, k) be an instance for the trivially perfect completion problem.

Lemma 5.2.3. *If a vertex v is not contained in any P_4 or C_4, then* $\operatorname{opt}(G - v) = \operatorname{opt}(G)$.

Proof. It is trivial that $\mathrm{opt}(G - v) \leq \mathrm{opt}(G)$. For the other direction, we show a stronger statement: any minimal solution E_+ to $G - v$ is also a solution to G. Let $\widehat{G} = G + E_+$. We verify that $\widehat{G}$ is a trivially perfect graph by showing that it satisfies Theorem 5.1.1(ii). If v is universal in G, hence also in $\widehat{G}$, then we are done; otherwise we show that $\widehat{G}$ is a trivially perfect graph if and only if a proper induced subgraph of $\widehat{G}$ is.

If G is not connected, then we can consider the only component that contains v. If G is connected but $G - v$ is not, then v has to be universal in G; otherwise, there is a P_4 containing v. In the last and the general case, v is not universal in G and $G - v$ is connected. We argue that at least one vertex in $N_G(v)$ is universal in $\widehat{G} - v$. Let u be any universal vertex of $\widehat{G} - v$. We are done if $u \in N_G(v)$, and henceforth we assume that $u \notin N_G(v)$. Since v is not in any P_4, the distance between v and u in G is at most two. Let $u' \in N_G(u) \cap N_G(v)$. From that v is not in any P_4 or C_4 it can be inferred that $N_G[u] \subset N_G[u']$ and $N_{G-v}[u] \subseteq N_{G-v}[u']$. By Lemma 5.2.2, $N_{\widehat{G}-v}[u] \subseteq N_{\widehat{G}-v}[u']$, and hence u' is also universal in $\widehat{G} - v$. In either case, we have found a vertex $x \in N_G(v)$ that is universal in $\widehat{G} - v$. By Proposition 5.2.1, $\widehat{G} - \{x, v\}$ is a minimal trivially perfect completion of $G - \{x, v\}$. Since the graph is finite, the claim follows. □

As a simple result of Lemma 5.2.3, we have the following reduction rule. In particular, all universal vertices of every component of G can be removed.

Rule 5.2.1. *If there is a vertex v that is not contained in any P_4 or C_4, then remove v.*

For each induced 4-path or 4-cycle $v_1v_2v_3v_4$, we call the missing edges $\{v_1, v_3\}$ and $\{v_2, v_4\}$ the *candidate edges* for this path or cycle. Clearly, any solution of a graph G contains at least one candidate edge of every P_4 or C_4; note that a P_4 has another missing edge, the addition of which merely turns the P_4 into a C_4.

Rule 5.2.2. *If uv is a candidate edge of $k+1$ or more P_4's and C_4's in G, then add the edge uv and decrease k by one.*

Now we show the safeness of this reduction rule.

Lemma 5.2.4. *Rule 5.2.2 is safe.*

Proof. Since each P_4 or C_4 of G has precisely two candidate edges, if a solution E_+ of G does not contain uv, then E_+ must contain the other candidate edge of each of the $k+1$ P_4's and C_4's (see an example in Figure 5.1), hence we have that $|E_+| > k$. □

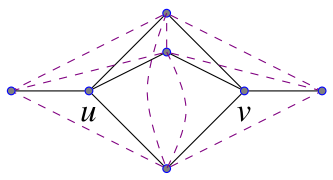


Figure 5.1: An example to show Lemma 5.2.4. The canditate edge uv are contained in three C_4's and six P_4's, if we do not add uv, then we need to add all violet dashed canditate edges.

5.3 A quadratic kernel

We summarize our kernelization algorithm for the trivially perfect completion problem in Figure 5.2.

Now we are ready to proof Theorem 1.4.4, we recall it here.

Theorem 5.3.1. *There is a $(2k^2+2k)$-vertex kernel for the trivially perfect completion problem.*

```
procedure reduce($G, k$)

1. if $k < 0$ then return a trivial no-instance;
2. if a vertex $v$ is not in any $P_4$ or $C_4$ then
       apply Rule 5.2.1 and return reduce($G, k$);
3. if $uv$ is a canditate edge of $k + 1$ or more $P_4$'s or $C_4$'s then
       apply Rule 5.2.2 and return reduce($G, k$);
4. if $|V(G)| \leq 2k^2 + 2k$ then return $(G, k)$;
5. else return a trivial no-instance.
```

Figure 5.2: The kernelization algorithm for Trivially Perfect Completion.

Proof. After applying Rule 5.2.2 and then Rule 5.2.1 exhaustively, we return (G, k) if $|V(G)| \leq 2k^2 + 2k$, or a trivial no-instance otherwise. We consider all the candidate edges of G. We say that two candidate edges are associated if they belong to the same P_4 or C_4; i.e., their ends are disjoint and together induce a P_4 or C_4. Since Rule 5.2.2 is not applicable, each candidate edge is associated with at most k candidate edges. On the other hand, of any two associated edges, one has to be in any solution of G. Thus, if (G, k) is a yes-instance, there can be at most $k^2 + k$ candidate edges. Since Rule 5.2.1 is not applicable, every vertex is in some P_4 or C_4, and hence is an end of a candidate edge. Thus, $|V(G)| \leq 2k^2 + 2k$ if (G, k) is a yes-instance. □

We now analyze the running time of this algorithm. When each time the algorithm calls itself in step 2, or 3, it removes at least one vertex from the graph. Therefore, the recursive calls can be made at most n times. On the other hand, each step clearly $O(n^4)$ time. Therefore, the algorithm returns in $O(n^5)$ time.

The analysis of the kernel in Theorem 5.3.1 is essentially the same as Buss and Goldsmith's kernelization algorithm for the vertex cover problem [21]. In a sense, we are looking for a vertex cover of an auxiliary graph in which each vertex corresponds to a candidate edge of G, and two vertices are adjacent if their corresponding edges are associated. We note that the same approach implies a simple $O(k^2)$-vertex kernel for the threshold completion problem, matching the result of Drange et al. [54]. The forbidden induced subgraphs of threshold graphs are $2K_2$, P_4, and C_4. The observation on the missing edges of a P_4 or C_4 is the same as above, while the four missing edges of a $2K_2$ can be organized as two pairs such that each solution has to contain at least one from each pair. However, we are not able to employ the $2k$-vertex kernels for vertex cover to directly derive a linear-vertex kernel for either of the two problems.

Before closing this section, let us mention some observations that might be of independent interest. The first is a simple corollary of Lemma 5.2.2.

Corollary 5.3.2. *If two vertices are true twins of a graph G, then they remain true twins of any minimal trivially perfect completion of G.*

A set M of vertices is a module if $N(M) = N(v) \setminus M$ for every $v \in M$. For example, a set of true twins is a module. Corollary 5.3.2 can be generalized to modules. For the last lemma, we use the fact that trivially perfect graphs are intersection graphs of nested intervals. (It can also be derived using the characterization by forbidden induced graphs.) A set of intervals representing an interval graph G is called an *interval representation* for G, where the interval for a vertex v is $I(v)$.

Lemma 5.3.3. *A module M of a graph G remains a module in any minimal trivially perfect completion $\widehat{G}$ of G.*

Proof. Let $E_+ = E(\widehat{G}) \setminus E(G)$. The claim follows from Corollary 5.3.2 when $\widehat{G}[M]$ is a clique: $\widehat{G}$ is also a minimal trivially perfect completion of G', where

G' is the graph obtained from G by adding edges to make M a clique. In the rest M is not a clique of $\widehat{G}$, hence not a clique of G.

Suppose for contradiction that M is not a module of $\widehat{G}$. Let U be the set of common neighbors of M in $\widehat{G}$. Since $\widehat{G}[M]$ is not a clique, $\widehat{G}[U]$ must be a clique. Moreover, $U \subset N_{\widehat{G}}(M)$. We take the leftmost endpoint ℓ and the rightmost endpoint r of $\bigcup_{v \in M} I(v)$. Note that $[\ell, r] \subseteq I(v)$ for every $v \in U$, and $I(x) \subseteq I(v)$ for every $x \in N_{\widehat{G}}(M) \setminus U$ and every $v \in U$. Let $p = r - \ell$, and we revise the intervals as follows. We increase each endpoint $\geq r$ by $p + 2$; and for each vertex $x \in N_{\widehat{G}}(M) \setminus U$, we set $I(x)$ to be $I(x) + p + 1$. (Informally speaking, we slide intervals for $N_{\widehat{G}}(M) \setminus U$ to the right so that they are disjoint from those for M.) We consider the graph G' represented by the revised intervals. It is easy to verify that these interval are still nested, and $E(\widehat{G}) \setminus E(G')$ is precisely the set of edges between M and $N_{\widehat{G}}(M) \setminus U$. Since M is a module of G, we have $N_G(M) \subseteq U$. Thus, $E(G) \subseteq E(G') \subset E(\widehat{G})$, which contradicts the minimality of $\widehat{G}$. This concludes the proof. □

Chapter 6

Split Graphs and Pseudo-split Graphs

Split graphs were defined and characterized by Földers and Hammer [149]. Recall that a graph is a split graph if its vertex set can be partitioned into a clique and an independent set. We use $C \uplus I$, where C being a clique and I an independent set, to denote a split partition of a split graph (see an example in Figure 6.1). The forbidden induced subgraphs of split graphs are $2K_2$, C_4, and C_5. From both the definition and the forbidden induced subgraphs ($\overline{C_4} = 2K_2$ and $\overline{C_5} = C_5$) we can see that the complement of a split graph is also a split graph. Thus, the split completion problem is polynomially equivalent to the split edge deletion problem. For the convenience of presentation, we work on the edge deletion problem.

Recall that a pseudo-split graph is either a split graph or a graph whose vertex set can be partitioned into a clique C, an independent set I, and a set S such that (1) S induces a C_5; (2) $C \subseteq N(v)$ for every $v \in S$; and (3) $I \cap N(v) = \emptyset$ for every $v \in S$. We use $C \uplus I \uplus S$ to denote a pseudo-split partition of a pseudo-split graph,

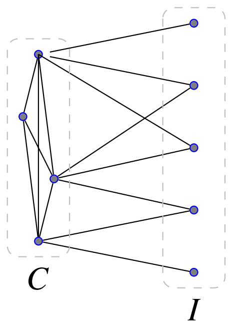


Figure 6.1: A split graph with a split partition $C \uplus I$.

where S may or may not be empty (see an example in Figure 6.2). If S is empty, then $C \uplus I$ is a split partition of the graph. Otherwise, the graph has a unique pseudo-split partition. The forbidden induced subgraphs of pseudo-split graphs are $2K_2$ and C_4. Similar as split graphs, the complement of a pseudo-split graph remains a pseudo-split graph. Thus, the completion problem and the edge deletion problem toward pseudo-split graphs are polynomially equivalent. In this section, we study the pseudo-split edge deletion problem.

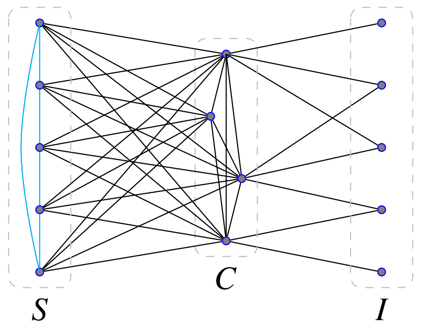


Figure 6.2: A pseudo-split graph with a pseudo-split partition $C \uplus I \uplus S$.

The class of pseudo-split graphs is a superclass of split graphs. In particular, split graphs are precisely C_5-free pseudo-split graphs. We can get a split graph from a pseudo-split graph that contains a C_5 by removing any vertex from the C_5 (a pseudo-split graph contains at most one C_5). Actually, the important part in many algorithms for graph modification problems to pseudo-split graphs is to dealing with the C_5.

For the split editing problem, Hammer and Simeone [89] have presented a polynomial-time algorithm to find a minimum solution to it. Suppose that the vertices are sorted by their degrees from the largest to the smallest, with ties broken arbitrarily. They showed that it suffices to find a certain number k, and make the first k vertices a clique and the rest an independent set. In other words, the solution comprises all the missing edges among the first k vertices and all the edges among other vertices. It does not seem to be easy, if plausible at all, to adapt this approach to solve the pseudo-split edge editing problem. See the graph in Figure 6.3 for an example, where the vertices are numbered by their degrees. The degree sequence is then

$$\langle 7, 5, 4, 4, 4, 4, 4, 4, 2 \rangle,$$

in which the five vertices in S are not consecutive.

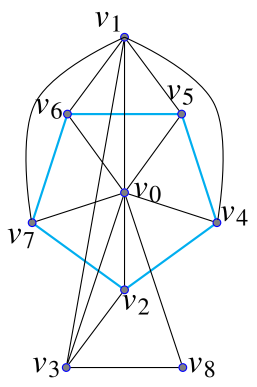


Figure 6.3: An example for the pseudo-split editing problem. The only optimal solution is to add the missing edge v_1v_2 and delete edges v_2v_3 and v_3v_8, with the pseudo-split partition $\{v_0, v_1\} \uplus \{v_3, v_8\} \uplus \{v_2, v_4, v_5, v_6, v_7\}$.

Drange [53] has show that the pseudo-split editing problem also can be soloved in polynomial time. The crucial observation is that if there is a minimum solution $E_{\pm}$ to the input graph G such that $G \triangle E_{\pm}$ contains a C_5, then this C_5 is contained in G and there is at most one missing edge between the clique set and the C_5, and at most one edge between the independent set and the C_5. Since there is at most one C_5 in a pseudo-split graph, we can solve the pseudo-split editing problem in polynomial time by check every C_5 in G.

In this section, we study the kernelization algorithms for both of the split edge deletion problem and the pseudo-split edge deletion problem. We consider the partition of the vertex set after applying an optimal solution. We observe that for most of the vertices we know to which side they have to belong. It is nevertheless not safe to directly delete these "decided" vertices. We thus work on the annotated version, where we mark certain vertices that have to be in the independent set. Guo [84] has proved that it is safe to remove a vertex that is not contained in any $2K_2$, C_4, or C_5. We show that a similar rule can be applied to annotated instances, and after its application, there can be $O(k^{1.5})$ vertices in a yes-instance. Finally, a simple step that removes the marks concludes the algorithm. Our kernel for the split edge deletion problem has only $O(k^{2.5})$ edges. With minor tweaks, our algorithm produces a kernel of the same size for the pseudo-split ($\{2K_2, C_4\}$-free graphs) edge deletion problem. A pseudo-split graph is either a split graph or a split graph plus a C_5. The first difficulty toward this adaptation is that it is not always safe to remove vertices not contained in any $2K_2$ or C_4. We get over this obstacle by observing that we can remove vertices not contained in any $2K_2$, C_4, or C_5. As we recycle the reduction rules for split edge deletion, only the arguments for their safeness need to be slightly revised.

6.1 Split edge deletion (completion)

Note that an instance (G, k) is a yes-instance for the split edge deletion problem if and only if there exists a partition $\{C, I\}$ of $V(G)$ such that C is a clique and $|E(I, I)| \leq k$; this is a split partition of $G - E(I, I)$. We call such a partition a *valid partition* of the instance (G, k) and use $C \uplus I$ to denote it, and we call C and I the clique set and independent set, respectively, in the valid partition. The problem is thus equivalent to finding a valid partition. We notice that some vertices can be easily decided to which side of a valid partition they should belong. For example, unless the instance is trivial, a simplicial vertex always belong to the independent set in any valid partition. Even after we know the destinations of these vertices, however, we cannot safely delete them. This brings us to the *annotated version* of the problem, where we mark certain vertices that can only be put into the independent set in a valid partition. We use (G, I_0, k) to denote such an annotated instance, where I_0 denotes the set of marked vertices. The original instance can be viewed as $(G, \emptyset, k)$, and a valid partition of an annotated instance (G, I_0, k) needs to satisfy the additional requirement that $I_0 \subseteq I$.

We can easily retrieve back an unannotated instance from an annotated instance. It suffices to add a small number of new vertices and make each of them adjacent to all other vertices but I_0.

Rule 6.1.1. *Let (G, I_0, k) be an annotated instance. Add a clique of $\sqrt{2k} + 1$ new vertices, and make each of them adjacent to all the vertices in $V(G) \setminus I_0$. Return the result as an unannotated instance.*

Now we show the safeness of Rule 6.1.1

Lemma 6.1.1. *Rule 6.1.1 is safe.*

Proof. Let K denote the clique of new vertices, and let (G', k) be the resulting instance. For any valid partition $C \uplus I$ of (G, I_0, k), the partition $(C \cup K) \uplus I$ is a valid partition of (G', k) because $C \subseteq V(G) \setminus I \subseteq N(x)$ for every $x \in K$. For a valid partition $C \uplus I$ of (G', k), if any vertex in I_0 is in C, then we must have $K \subseteq I$. Since K is a clique of order $\sqrt{2k} + 1$, we have $|E(I, I)| > k$, which contradicts the validity of the partition. □

The aforementioned observation on simplicial vertices is formalized by the following rule.

Rule 6.1.2. *Let v be a simplicial vertex in $V(G) \setminus I_0$. If $|E(G - (N[v] \setminus I_0))| \leq k$, then return a trivial yes-instance. Otherwise, add v to I_0.*

Now we show the safeness of Rule 6.1.2

Lemma 6.1.2. *Rule 6.1.2 is safe.*

Proof. In the first case, $(N[v] \setminus I_0) \uplus (V(G) \setminus N[v] \cup I_0)$ is a valid partition. Otherwise, we show by contradiction that $v \in I$ in any valid partition $C \uplus I$ of (G, I_0, k). Since C is a clique, if $v \in C$, then $C \subseteq N[v] \setminus I_0$. Thus, $E(G - (N[v] \setminus I_0)) \subseteq E(I, I)$, but then $|E(I, I)| > k$, contradicting the validity of the partition. □

We construct a modulator M as follows. We greedily find a maximal packing of vertex-disjoint $2K_2$'s, C_4's, and C_5's. Let M be the set of vertices in all subgraphs we found. We can terminate the algorithm by returning a trivial no-instance if we have found more than k vertex-disjoint forbidden induced subgraphs from G. Henceforth, we may assume that $|M| \leq 5k$, and we fix a split partition $C_M \uplus I_M$

of $G - M$. The following simple observation enables us to know the destinations of more vertices.

Lemma 6.1.3. *For any valid partition $C \uplus I$ of (G, k), if one exists, we have*

(i) $|I_M \cap C| \leq 1$*; and*

(ii) $|C_M \cap I| \leq \sqrt{2k}$.

Proof. The first assertion follows from that I_M is an independent set and C is a clique. The second assertion holds because

$$k \geq |E(I, I)| \geq |E(C_M \cap I, C_M \cap I)| = \binom{|C_M \cap I|}{2}. \qquad \square$$

We say that a vertex is a *c-vertex*, respectively, an *i-vertex*, if it is in C, respectively, in I, for any valid partition $C \uplus I$ of (G, I_0, k). Clearly, every vertex that has more than $k + 1$ neighbors in I_M is a c-vertex, while the following are i-vertices:

- every vertex with more than $\sqrt{2k}$ non-neighbors in C_M; and
- every vertex nonadjacent to a c-vertex.

We can indeed delete all the c-vertices, as long as we keep their non-neighbors marked. Note that after obtaining the initial split partition $C_M \uplus I_M$ of $G - M$, we do not need to maintain the invariant that M is a modulator, though we do maintain that C_M is a clique and that I_M is an independent set throughout. During our algorithm, we maintain $\{M, C_M, I_M, I_0\}$ as a partition of $V(G)$. Therefore, whenever we mark a vertex, we remove it from the set that originally contains it, and move it to I_0.

Rule 6.1.3. *Let (G, I_0, k) be an annotated instance.*

(i) Mark every vertex that has more than $\sqrt{2k}$ non-neighbors in C_M.

(ii) If a vertex v has more than $k+1$ neighbors in $I_M \cup I_0$, then mark every vertex in $V(G) \setminus N[v]$ and delete v.

Now we show the safeness of Rule 6.1.3.

Lemma 6.1.4. *Rule 6.1.3 is safe.*

Proof. Let I'_0 denote the set of marked vertices after the reduction. It is trivial that if the resulting instance of (i) is a yes-instance, then the original is also a yes-instance. For (ii), any valid partition $C' \uplus I'$ of $(G - v, I'_0, k)$ can be extended to a valid partition $(C' \cup \{v\}) \uplus I'$ of (G, I_0, k) because $C' \subseteq V(G) \setminus I'_0 \subseteq N[v]$.

For the other direction, let $C \uplus I$ be any valid partition of (G, I_0, k). (i) Since C is a clique, $C_M \setminus N(v) \subseteq I$ for every $v \in C$. By Lemma 6.1.3(ii), if $|C_M \setminus N(v)| > \sqrt{2k}$ for some vertex v, then v has to be in I. Thus, we get that $C \uplus I$ is also a valid partition of the new instance (G, I'_0, k). (ii) By Lemma 6.1.3(i), we have $|I_M \setminus I| \leq 1$. As $I_0 \subseteq I$ and $|N(v) \cap (I_M \cup I_0)| > k + 1$, there are at least $k + 1$ edges between v and I. Since $|E(I, I)| \leq k$, we must have $v \in C$. Moreover, since C is a clique, we get that $C \subseteq N[v]$, and every vertex nonadjacent to v has to be in I. This justifies the marking of $V(G) \setminus N[v]$. Clearly, the partition $(C \setminus \{v\}) \uplus I$ is a valid partition of $(G - v, I'_0, k)$. □

The next rule is straightforward: since I_0 has to be in the independent set of a valid partition, every solution contains all the edges in $E(I_0, I_0)$.

Rule 6.1.4. *Let (G, I_0, k) be an annotated instance. Remove all the edges in $E(I_0, I_0)$, and decrease k accordingly.*

Now we show the safeness of Rule 6.1.4.

Lemma 6.1.5. *Rule 6.1.4 is safe.*

Proof. By the definition of the annotated instance, any solution E_- of (G, I_0, k) contains all the edges in $E(I_0, I_0)$. Moreover, the edge set $E_- \setminus E(I_0, I_0)$ is a solution to $G - E(I_0, I_0)$, and its size is at most $k - |E(I_0, I_0)|$. On the other hand, if $(G - E(I_0, I_0), k - |E(I_0, I_0)|)$ is a yes-instance, then any solution of this instance, together with $E(I_0, I_0)$, makes a solution of (G, I_0, k) of size at most k. □

Once there are no edges among vertices in I_0, we can replace I_0 with another independent set as long as we keep track of the number of edges between every vertex $v \in V(G) \setminus I_0$ and I_0. The following rule reduces the cardinality of I_0.

Rule 6.1.5. *Let (G, I_0, k) be an annotated instance where I_0 is an independent set. Introduce p new vertices v_1, v_2, ..., v_p, where $p = \max_{v \in V(G)} |N(v) \cap I_0|$. For each vertex $x \in N(I_0)$, make x adjacent to v_1, ..., $v_{|N(x) \cap I_0|}$. Remove all vertices in I_0, and mark the set of new vertices.*

Note that if Rule 6.1.3 is not applicable, then $p \leq k + 1$. Instead of proving the safeness of Rule 6.1.5, we prove a stronger statement.

Lemma 6.1.6. *Let (G, I_0, k) and (G', I'_0, k) be two annotated instances where $G - I_0 = G' - I'_0$ and both I_0 and I'_0 are independent sets. If $|N_G(x) \cap I_0| = |N_{G'}(x) \cap I'_0|$ for every $x \in V(G) \setminus I_0$, then (G, I_0, k) is a yes-instance if and only if (G', I'_0, k) is a yes-instance.*

Proof. Let $\{C, I\}$ be a partition of $V(G)$. We show that $C \uplus I$ is a valid partition of (G, I_0, k) if and only if $C \uplus ((I \setminus I_0) \cup I'_0)$ is a valid partition of (G', I'_0, k). Note that

$$|E(I \setminus I_0, I_0)| = \sum_{x \in I \setminus I_0} |N_G(x) \cap I_0| = \sum_{x \in I \setminus I'_0} |N_{G'}(x) \cap I'_0| = |E(I \setminus I'_0, I'_0)|.$$

Since $G - I_0 = G' - I'_0$, and since there is no edge in $G[I_0]$ or $G'[I'_0]$, the claim follows. □

Let us recall an important observation of Guo [84].

Lemma 6.1.7 ([84]). *If a vertex v is not contained in any $2K_2$, C_4, or C_5, then* $\operatorname{opt}(G - v) = \operatorname{opt}(G)$.

Both Guo [84] and Ghosh et al. [75] used a rule derived from this observation to delete vertices, and this is their only rule that removes vertices from the graph. We may show that the same rule indeed works for our annotated instances, for which however we have to go through the original argument of [84]. We note that if a vertex v in I_0 is adjacent to two vertices u and w with $uw \notin E(G)$, then any solution has to contain at least one of edges uv and vw (since u and w cannot be both in the clique). We say that an induced P_3 is I_0-*centered* if the degree-two vertex of this P_3 is from I_0. In a sense, I_0-centered P_3's are "minimal forbidden structures" for our annotated instances. Accordingly, a C_4 or C_5 involving a vertex from I_0 is no longer minimal. In summary, the "minimal forbidden structures" are C_4's and C_5's in $G - I_0$, all $2K_2$'s, and I_0-centered P_3's. Note that a "minimal forbidden structure" intersecting I_0 has to be a $2K_2$ or an I_0-centered P_3, and this gives another explanation of the correctness of Lemma 6.1.6, which exchanges these two kinds of "minimal forbidden structures" with each other. The following rule can be viewed as the annotated version of the rule of Guo [84], and its safeness can be argued using Lemma 6.1.7.

Rule 6.1.6. *Let (G, I_0, k) be an annotated instance where I_0 is an independent set, and let v be a vertex in C_M. If v is not contained in any $2K_2$ or any I_0-centered P_3, and every C_4 and C_5 that contains v intersects I_0, then remove v from G.*

Now we prove the safeness of Rule 6.1.6.

Lemma 6.1.8. *Rule 6.1.6 is safe.*

Proof. Suppose that v is a vertex not contained in any $2K_2$ or any I_0-centered P_3, and every C_4 and C_5 that contains v intersects I_0. We show that (G, I_0, k) is a yes-instance if and only if $(G - v, I_0, k)$ is a yes-instance, by establishing a sequence of equivalent instances. For each edge $xy \in E(G)$ with $x \in I_0$ and $y \in V(G) \setminus I_0$, we introduce a new vertex v_{xy} and make it adjacent to y. Remove all vertices in I_0, and let I'_0 denote the set of new vertices (see Figure 6.4). Let (G', I'_0, k) denote the resulting instance. The equivalence between (G', I'_0, k) and (G, I_0, k) follows from Lemma 6.1.6. Then let (G'', k) denote the instance obtained by applying Rule 6.1.1 to (G', I'_0, k), with K being the added clique of size $\sqrt{2k} + 1$.

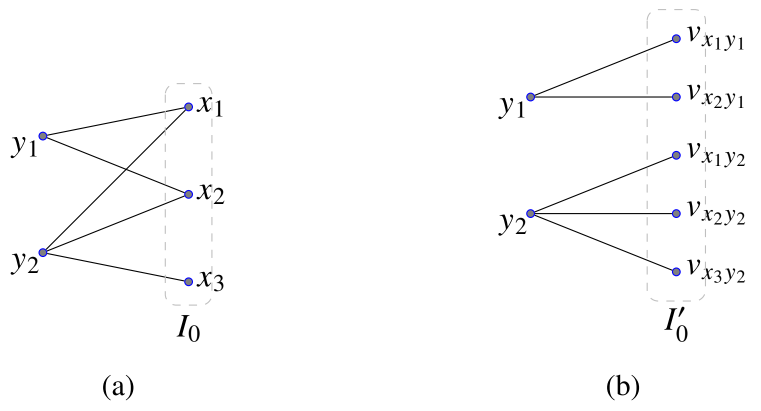


Figure 6.4: Illustration of Lemma 6.1.8. The transformation from G to G'.

We argue that v is not contained in any $2K_2$, C_4, or C_5 of G''. Suppose for contradiction that there is a set $F \subseteq V(G'')$ that contains v and induces a $2K_2$, C_4, or C_5 in G''. Since neither the transformation from G to G' nor the transformation from G' to G'' makes any change to $V(G) \setminus I_0$, this set induces the same subgraph in G and G''. Thus, the set $F \not\subseteq V(G) \setminus I_0$. Moreover, since every vertex in K is universal in $G'' - I'_0$ and $\{V(G) \setminus I_0, K, I'_0\}$ is a partition of $V(G'')$, it follows that $F \cap I'_0$ is not empty. Note that every vertex in I'_0 has only one neighbor in G'', we can conclude that $G''[F]$ must be a $2K_2$ and $F \cap K = \emptyset$. But then v is contained in either a $2K_2$ or an I_0-centered P_3 in G, a contradiction.

It then follows from Lemma 6.1.7 that (G'', k) is equivalent to $(G'' - v, k)$. To see the equivalence between $(G'' - v, k)$ and $(G - v, I_0, k)$, we apply the reversed operations from G to G''. We first use Rule 6.1.3, applied to $(G'' - v, \emptyset, k)$, to mark all vertices in I'_0, then use Lemma 6.1.6 to replace I'_0 by I_0, and finally remove vertices in K. The resulting graph is precisely $G - v$. We can thus conclude the proof. □

We call an annotated instance *reduced* if none of Rules 6.1.2–6.1.6 is applicable to this instance. The following lemma bounds the cardinalities of C_M and I_M in a reduced instance.

Lemma 6.1.9. *If a reduced instance (G, I_0, k) is a yes-instance, then $|C_M| \leq 3k\sqrt{2k}$ and $|I_M| \leq k + 1$.*

Proof. Let E_- be any solution to (G, I_0, k) with at most k edges. Since Rule 6.1.6 is not applicable, every vertex in C_M is contained in some $2K_2$ or I_0-centered P_3, or some C_4 or C_5 in the subgraph $G - I_0$. Any of these structures must contain an edge in E_-. Therefore, to bound $|C_M|$, it suffices to count how many vertices in C_M

can form a $2K_2$ or I_0-centered P_3, or a C_4 or C_5 in $G - I_0$ with an edge $xy \in E_-$.

- If a vertex $v \in C_M$ is in a $2K_2$ with the edge xy, then either $v \in \{x, y\}$ or v is adjacent to neither x nor y. In the first case, no other vertex in C_M can occur in any $2K_2$ with xy. Since $xy \in E(G)$, at least one of its endpoints is not in I_0 (Rule 6.1.4). This vertex has at most $\sqrt{2k}$ non-neighbors in C_M. Therefore, the total number of vertices in C_M that can occur in any $2K_2$ with xy is at most $\sqrt{2k}$.

- If xy is an edge in any I_0-centered P_3, then precisely one of x and y is in I_0. Assume without loss of generality $x \in I_0$. If a vertex $v \in C_M$ is in an I_0-centered P_3 with the edge xy, then either $v = y$, or v is not adjacent to y. Since $y \notin I_0$, it has at most $\sqrt{2k}$ non-neighbors in C_M. Thus, the total number of vertices in C_M that can occur in any I_0-centered P_3 containing xy is at most $\sqrt{2k} + 1$.

- If a vertex $v \in C_M$ forms a C_4 or C_5 in $G - I_0$ with the edge xy, then v is adjacent to at most one of x and y. Since this C_4 or C_5 is in $G - I_0$, each of x and y has at most $\sqrt{2k}$ non-neighbors in C_M. Thus, the total number of vertices in C_M that can occur in such a C_4 or C_5 is at most $2\sqrt{2k}$.

Noting that an edge cannot satisfy the conditions of both the second ($|\{x, y\} \cap I_0| = 1$) and third ($|\{x, y\} \cap I_0| = 0$) categories, we can conclude $|C_M| \leq k(\sqrt{2k} + 2\sqrt{2k}) = 3k\sqrt{2k}$.

Since Rule 6.1.2 is not applicable, no vertex in I_M is simplicial. Suppose that $C \uplus I$ is a valid partition of (G, k). Since C is a clique, for each vertex $v \in I_M \cap I$, at least one neighbor of v is in I. Therefore, each vertex $v \in I_M \cap I$ is incident to an edge in the solution $E(I, I)$. Noting that I_M is an independent set, we have $k \geq |I_M \cap I| \geq |I_M| - 1$, where the second inequality follows from Lemma 6.1.3(i). Thus, we can conclude $|I_M| \leq k + 1$, and this concludes this proof. □

Note that the application of Rule 6.1.1 is different from the other ones. The application of one of Rules 6.1.2–6.1.6 may trigger the applicable of another. After the application of Rule 6.1.1, the instance is no longer annotated, and we will not go back to check the other rules.

6.2 An improved kernel

We summarize the kernelization algorithm in Figure 6.5.

Input: an instance (G, k) of the split edge deletion problem.
Output: an equivalent instance (G', k') with $|V(G')| = O(k'^{1.5})$.

```
1.   $I_0 \leftarrow \emptyset$;
1.   $M \leftarrow$ a maximal packing of vertex-disjoint $2K_2$'s, $C_4$'s, and $C_5$'s;
2.   if $|M| > 5k$ then return a trivial no-instance;
3.   if $k < 0$ then return a trivial no-instance;
4.   for each simplicial vertex $v \in V(G) \setminus I_0$ do (Rule 6.1.2)
         if $|E(G - (N[v] \setminus I_0))| \leq k$ then return a trivial yes-instance;
         else $I_0 \leftarrow I_0 \cup \{v\}$;
5.   remove c-vertices and mark i-vertices (Rule 6.1.3);
6.   if $E(I_0, I_0) \neq \emptyset$ then
         remove edges in $E(I_0, I_0)$ and decrease $k$ (Rule 6.1.4);
7.   merge $I_0$ into $\leq k$ vertices (Rule 6.1.5);
8.   remove vertices in $C_M$ not contained in certain structures (Rule 6.1.6);
9.   if any of Rules 6.1.2–6.1.4 and 6.1.6 made a change then goto 3;
10.  if $|C_M| + |I_M| > 3k\sqrt{2k} + k + 1$ then return a trivial no-instance;
11.  add $\sqrt{2k} + 1$ new vertices and remove all marks (Rule 6.1.1);
12.  return $(G, k)$.
```

Figure 6.5: A summary of our kernelization algorithm for Split Edge Deletion.

Now we are ready to prove Theorem 1.4.5 and Theorem 1.4.6, we recall them here.

Theorem 6.2.1. *The split edge deletion problem has a kernel with at most $O(k^{1.5})$ vertices.*

Proof. We use the algorithm described in Figure 6.5. The first two steps build the modulator, and their correctness follows from that any solution contains at least one edge of each forbidden induced subgraph of G. Step 3 is obviously correct. Steps 4–8 follow from the safeness of the rules; so is step 11. The correctness of step 10 is ensured by Lemma 6.1.9.

The cardinality of M is at most $5k$, and it never increases during the algorithm. After step 7, we have $|I_0| \leq k$. We have bounded the cardinalities of C_M and I_M in Lemma 6.1.9. Step 11 increases $|C_M|$ by $\sqrt{2k}+1$. Putting them together, we have

$$|V(G)| \leq 5k + k + (3k\sqrt{2k} + \sqrt{2k} + 1) + k + 1 = O(k^{1.5}).$$

It is easy to verify that each reduction rule can be checked and applied in polynomial time. To see that the algorithm runs in polynomial time, note that if any of Rules 6.1.2–6.1.4 and 6.1.6 made a change to the instance, then either k decreases by one (Rule 6.1.4), or the cardinality of $V(G) \setminus I_0$ decreases by one (the other three rules). □

Since the class of split graphs is self-complementary, our algorithm also implies a kernel for the split completion problem. This kernel actually has fewer edges than the one for Split Edge Deletion.

Theorem 6.2.2. *The split completion problem has a kernel with at most $O(k^{1.5})$ vertices and $O(k^{2.5}) edges$.*

Proof. Let (G, k) be the input instance of the split completion problem. We can either take the complement of the input graph and consider it as an instance of the split edge deletion problem, or run the "complemented versions" of the rules. In the final result, we have an independent set of at most $O(k\sqrt{k})$ vertices, and at most $O(k)$ other vertices. The claim then follows. □

6.3 Pseudo-split edge deletion (completion)

AS we mentioned before, split graphs are precisely C_5-free pseudo-split graphs. Note that a pseudo-split graph contains at most one C_5. In case that a pseudo-split graph does contain a C_5, removing any vertex from the C_5 leaves a split graph. Another way to derive a split subgraph from a pseudo-split graph is by removing any two consecutive edges from the C_5. Thus, if we use $\text{sed}(G)$ to denote the size of the smallest edge set E_- such that $G - E_-$ is a split graph, then

$$\text{opt}(G) \leq \text{sed}(G) \leq \text{opt}(G) + 2.$$

Moreover, we have $\text{opt}(G) = \text{sed}(G)$ if and only if there is a minimum solution E_- of G (for the pseudo-split edge deletion problem) such that $G - E_-$ is a split graph.

We say that a partition $C \uplus I \uplus S$ of the vertex set of the input graph G is a *valid partition* of the instance (G, k) if there exists a set E_- of at most k edges such that $C \uplus I \uplus S$ is a pseudo-split partition of $G - E_-$.

Proposition 6.3.1. *Let G be a graph with $\text{opt}(G) < \text{sed}(G)$, let E_- be a minimum solution to G, and let $C \uplus I \uplus S$ be the pseudo-split partition of $G - E_-$. Then the subgraph $G[S]$ is a C_5, and no vertex in I forms a triangle with two vertices in S in G.*

Proof. Since $\text{opt}(G) < \text{sed}(G)$, the set S cannot be empty. Let $G' = G - E_-$, and let $v_1v_2v_3v_4v_5$ be the cycle of $G'[S]$. We first argue that $G[S]$ is a C_5. Suppose otherwise, then there is a chord of the cycle $G'[S]$, say v_1v_3, in E_-. We take $E'_- = (E_- \setminus \{v_1v_3\}) \cup \{v_4v_5\}$. Note that $G - E'_-$ is a split graph, as evidenced by the split partition $(C \cup \{v_1, v_2, v_3\}) \uplus (I \cup \{v_4, v_5\})$ (see Figure 6.6). But $|E'_-| = |E_-|$ contradicts $\text{opt}(G) < \text{sed}(G)$.

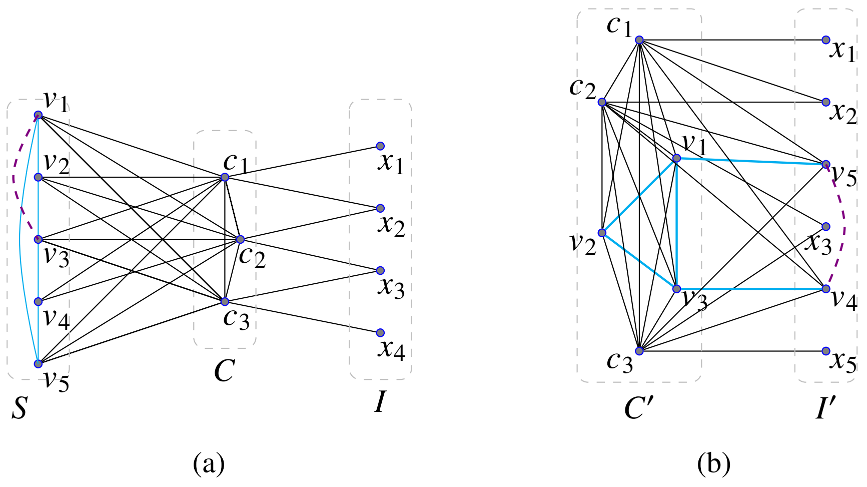


Figure 6.6: Illustration for Proposition 6.3.1.

For the second part, suppose for contradiction that there is a triangle of G containing a vertex $u \in I$ and two vertices in S. We have seen that $G[S]$ is a C_5. We may assume that the triangle is uv_1v_2. We take $E'_- = (E_- \setminus \{uv_1, uv_2\}) \cup \{v_3v_4, v_4v_5\}$. Note that $G - E'_-$ is a split graph, as evidenced by the split partition $(C \cup \{v_1, v_2\}) \uplus (I \cup \{v_3, v_4, v_5\})$. But $|E'_-| = |E_-|$, which contradicts $\text{opt}(G) <$

sed(G). □

For the pseudo-split edge deletion problem, one may expect a proposition similar as Lemmas 5.2.3 and 6.1.7; i.e., it is safe to remove vertices not in any $2K_2$ or C_4. As shown in Figure 6.7, this is however not true. This graph contains no $2K_2$, and the only C_4 is $v_1v_2v_3v_4$. The deletion of any edge from this cycle introduces a new $2K_2$, e.g., $\{v_1v_4, v_2v_6\}$ after v_1v_2 deleted. On the other hand, $\text{opt}(G - v_6) = 1$ because it suffices to delete either v_1v_2 or v_2v_3. We manage to show that if a vertex v is not in any $2K_2$, C_4, or C_5, then it is safe to remove v. This is sufficient for our algorithm.

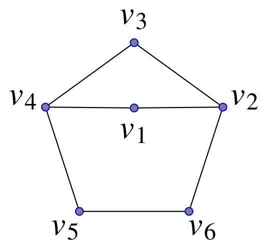


Figure 6.7: $\text{opt}(G) = 2$, while $\text{opt}(G - v_6) = 1$.

Lemma 6.3.2. *If a vertex v is not contained in any $2K_2$, C_4, or C_5, then* $\text{opt}(G-v) = \text{opt}(G)$.

Proof. Let $G' = G - v$. It is trivial that any solution to G contains a solution to G', hence $\text{opt}(G') \leq \text{opt}(G)$. Let E_- be a minimum solution to G'. We have nothing to show if $G - E_-$ is a pseudo-split graph as well. In the rest of the proof, $G - E_-$ is not a pseudo-split graph. We take a pseudo-split partition $C \uplus I \uplus S$ of $G' - E_-$. If $S = \emptyset$, then $G' - E_-$ is a split graph. Since every split graph is a pseudo-split graph, E_- is also a minimum split edge deletion set of $G - v$. By

Lemma 6.1.7, there is a set E'_- such that $|E'_-| = |E_-|$ and $G - E'_-$ is a split graph. Thus, $\text{opt}(G) \le |E'_-| = \text{opt}(G - v)$.

Now that $S \ne \emptyset$, we may assume without loss of generality that (1) $G[S]$ is a C_5, and (2) $|S \cap N_G(x)| \le 2$ for every vertex $x \in I$; otherwise, by Proposition 6.3.1, we can find another solution E'_- of G' such that $|E'_-| = |E_-|$ and $G' - E'_-$ is a split graph, and then we are in the previous case. Under these assumptions we show that the vertex v is either adjacent to all vertices in $C \cup S$, or nonadjacent to any vertex in $I \cup S$. Accordingly, we can get that either $(C \cup \{v\}) \uplus I \uplus S$ or $C \uplus (I \cup \{v\}) \uplus S$ is a pseudo-split partition of $G - E_-$, and hence $G - E_-$ is also a pseudo-split graph.

Let us start from the adjacency between v and S. If v is adjacent to only one vertex in S, or two or three consecutive vertices on the C_5, then v is contained in a $2K_2$. On the other hand, if v is adjacent to four vertices in S, or two or three non-consecutive vertices on the C_5, then v is contained in a C_4. See Figure 6.8 for illustration. Therefore, we have that v is adjacent to either all or none of the vertices in S. If $S \subseteq N_G(v)$, then $C \subseteq N_G(v)$ as well: v, a vertex $x \in C \setminus N_G(v)$, and two nonadjacent vertices in S would induce a C_4. Now that $N_G(v) \cap S = \emptyset$, we are done if $N_G(v) \cap I = \emptyset$. Suppose otherwise, and let u be any vertex in $N_G(v) \cap I$. By assumption (2), we have $|S \cap N_G(u)| \le 2$. But then an edge in $G[S]$ of which both endpoints nonadjacent to u form an induced $2K_2$ with uv in G. This concludes the proof. □

To adapt the algorithm in Figure 6.5 for the pseudo-split edge deletion problem, we only need to conduct very minor adjustments. We use the same modulator M as the previous section, i.e., vertices of a maximal packing of vertex-disjoint $2K_2$'s, C_4's, and C_5's. Recall that a pseudo-split graph contains at most one C_5. Thus, if

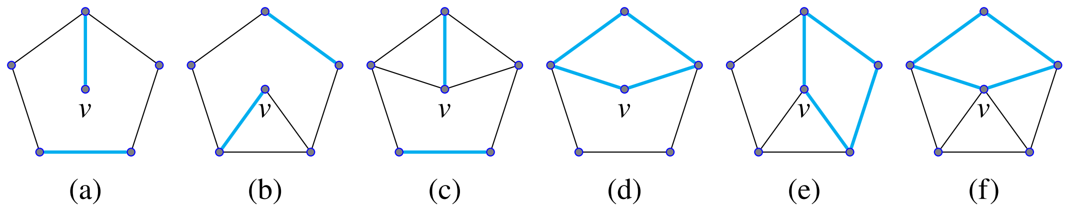


Figure 6.8: Illustration for the proof of Lemma 6.3.2. There is a $2K_2$ (a, b, c) or C_4 (d, e, f), shown as thick lines, containing v.

we have found p vertex-disjoint C_5's from G, then we need to break at least $p-1$ of them.

Lemma 6.3.3. *If (G, k) is a yes-instance, then $|M| \leq 4k+5$.*

Proof. Let E_- be a minimum solution to G. Suppose that the numbers of vertex-disjoint $2K_2$'s, C_4's, and C_5's we have put into M are p, q, and r, respectively. In each $2K_2$ or C_4, at least one edge needs to be in E_-. At most one C_5 can be disjoint from E_-, and on each of other C_5's, at least two edges are in E_-. Thus, we have $k \geq |E_-| \geq p+q+2(r-1)$, and $|M| = 4p+4q+5r = 4p+4q+5(r-1)+5 \leq 4k+5$. □

As a result, if $|M| > 4k+5$, then (G, k) must be a no-instance (step 2 of the algorithm). Again, we start from a split partition $C_M \uplus I_M$ of $G - M$, and we work on the annotated version of the problem. In an annotated instance (G, I_0, k), the set I_0 of marked vertices can only be put into the independent set I in a valid partition. We use the same rules as we have used for the split edge deletion problem. We now verify that all of them remain safe for the pseudo-split edge deletion problem. The first is simple.

Rule 6.3.1. *Let (G, I_0, k) be an annotated instance. Add a clique of $\sqrt{2k}+1$ new vertices, and make each of them adjacent to all the vertices in $V(G) \setminus I_0$. Return* *as an unannotated instance.*

Now we show the safeness of Rule 6.3.1.

Lemma 6.3.4. *Rule 6.3.1 is safe.*

Proof. Let K denote the clique of new vertices, and let (G', k) be the resulting instance. For any valid partition $C \uplus I \uplus S$ of (G, I_0, k), the partition $(C \cup K) \uplus I \uplus S$ is a valid partition of (G', k) because $C \subseteq V(G) \setminus I \subseteq N(x)$ and $S \subseteq V(G) \setminus I \subseteq N(x)$ for every vertex $x \in K$. Let E_- be a set of at most k edges such that $G' - E_-$ is a pseudo-split graph and $C \uplus I \uplus S$ is a pseudo-split partition of $G' - E_-$. If any vertex in I_0 is in $C \cup S$, then we must have $K \subseteq I$. But then $|E_-| > k$, which contradicts the validity of the partition. □

Rule 6.1.2 was safe for the split edge deletion problem because a vertex is either in C or I. For the pseudo-split edge deletion problem, we need to take care of the possibility that a simplicial vertex is in S.

Lemma 6.3.5. *Let (G, I_0, k) be an instance of the annotated version of the pseudo-split edge deletion problem, and let v be a simplicial vertex of G. If there exists a valid partition $C \uplus I \uplus S$ with $v \in S$, then there exists another valid partition $C' \uplus I' \uplus S'$ with $S' = \emptyset$.*

Proof. Since $v \in S$, the set S is not empty. Since v is simplicial, $G[S]$ is not a C_5. The statement follows from Proposition 6.3.1. □

As a result, for any simplicial vertex v, it suffices to look for a valid partition $C \uplus I \uplus S$ with $v \in C \cup I$. If $v \in C$, then $S = \emptyset$; since $C \cup S$ is a subset of $N[v]$, it is a clique, and then $(C \cup S) \uplus I$ is a partition with a smaller solution. Thus, it remains safe.

Rule 6.3.2. *Let v be a simplicial vertex in $V(G) \setminus I_0$. If $|E(G - (N[v] \setminus I_0))| \leq k$, then return a trivial yes-instance. Otherwise, add v to I_0.*

Now we show the safeness of Rule 6.3.2.

Lemma 6.3.6. *Rule 6.3.2 is safe.*

Proof. If $|E(G-(N[v] \setminus I_0))| \leq k$, then $(N[v] \setminus I_0) \uplus (V(G) \setminus N[v] \cup I_0)$ is a valid partition. Otherwise, we show that there is a minimum solution E_- to (G, I_0, k) such that $C \uplus I \uplus S$ is a pseudo-split partition of $G - E_-$ and $v \in I$. Suppose for contrary that $v \notin I$, then by Lemma 6.3.5, we can suppose that v is in C. Since C is a clique, then $C \subseteq N[v] \setminus I_0$. Since every vertex in S is adjacent to all vertices in C, then $S \subseteq N[v] \setminus I_0$. Thus, we have $E(G-(N[v] \setminus I_0)) \subseteq E_-$, but then $|E_-| > k$, contradicting the validity of the partition. □

Lemma 6.3.7. *Let $C \uplus I \uplus S$ be a valid partition of (G, k), if one exists, we have*

(i) $|I_M \cap C| \leq 1$*;*

(ii) $|I_M \cap S| \leq 2$*; and*

(iii) $|C_M \cap I| \leq \sqrt{2k}$*.*

Proof. The first assertion follows from that I_M is an independent set and C is a clique. The second assertion holds because a C_5 does not contain an independent set of order three. The last assertion holds because

$$k \geq |E(I, I)| \geq |E(C_M \cap I, C_M \cap I)| = \binom{|C_M \cap I|}{2}.$$ □

We say that a vertex is a *c-vertex*, respectively, an *i-vertex*, if it is in C, respectively, in I, for any valid partition $C \uplus I \uplus S$ of (G, I_0, k) (we do not consider vertices in S since there are only five such vertices).

Lemma 6.3.8. *If a vertex v has more than $k+3$ neighbors in I_M, then v is a c-vertex.*

Proof. Let E_- be a set of at most k edges such that $G - E_-$ is a pseudo-split graph and $C \uplus I \uplus S$ is a pseudo-split partition of $G - E_-$. Suppose for contrary that v is not a c-vertex, then it is in I or S. By Lemma 6.3.7 (i) and (ii), there are at least $k + 1$ neighbors of v are in I, then $|E_-| > k$, which is a contradiction. □

Lemma 6.3.9. *The following are i-vertices:*

- *every vertex nonadjacent to a c-vertex; and*
- *every vertex with more than $\sqrt{2k} + 1$ non-neighbors in C_M.*

Proof. The first assertion follows from the definition of the pseudo-split graphs. Now we proof the second assertion. Let v be a vertex with more than $\sqrt{2k} + 1$ non-neighbors in C_M, and let E_- be a set of at most k edges such that $G - E_-$ is a pseudo-split graph and $C \uplus I \uplus S$ is a pseudo-split partition of $G - E_-$. Suppose for contrary that $v \notin I$, then v is in C or S. If $v \in C$, then all non-neighbors of v in C_M are in I, then we have $|C_M \cap I| > \sqrt{2k}$, which contradicts Lemma 6.3.7 (iii). If $v \in S$, then there are at most two non-neighbors of v in C_M that are in S, then all but one edges in $G[(V(G) \setminus N[v]) \cap C_M]$ are in E_-, then $|E_-| > k$, a contradiction. □

We can indeed delete all the c-vertices, as long as we keep their non-neighbors marked. Note that after obtaining the initial split partition $C_M \uplus I_M$ of $G - M$, we do not need to maintain the invariant that M is a modulator, though we do maintain that C_M is a clique and that I_M is an independent set throughout. During our algorithm, we maintain $\{M, C_M, I_M, I_0\}$ as a partition of $V(G)$. Therefore, whenever we mark a vertex, we move it to I_0.

Rule 6.3.3. *Let (G, I_0, k) be an annotated instance.*

(i) Mark every vertex that has more than $\sqrt{2k} + 1$ non-neighbors in C_M.

(ii) If a vertex v has more than $k+3$ neighbors in $I_M \cup I_0$, then mark every vertex in $V(G) \setminus N[v]$ and delete v.

Now we show the safeness of Rule 6.3.3.

Lemma 6.3.10. *Rule 6.3.3 is safe.*

Proof. Let I_0' denote the set of marked vertices after the reduction. It is trivial that if the resulting instance of (i) is a yes-instance, then the original is also a yes-instance. For (ii), any valid partition $C' \uplus I' \uplus S'$ of $(G - v, I_0', k)$ can be extended to a valid partition $(C' \cup \{v\}) \uplus I' \uplus S'$ of (G, I_0, k) because $C' \subseteq V(G) \setminus I_0' \subseteq N[v]$ and $S' \subseteq V(G) \setminus I_0' \subseteq N[v]$.

For the other direction, let E_- be a set of at most k edges such that $G - E_-$ is a pseudo-split graph and $C \uplus I \uplus S$ is a pseudo-split partition of $G - E_-$ where $I_0 \subseteq I$. (i) Since $|E_-| \leq k$, then $|C_M \setminus N(v)| \leq \sqrt{2k}$ for every $v \in C$ and $|C_M \setminus N(v)| \leq \sqrt{2k} + 1$ for every $v \in S$. Therefore, if $|C_M \setminus N(v)| > \sqrt{2k} + 1$ for some vertex v, then v has to be in I. Thus, $C \uplus I \uplus S$ is also a valid partition of the new instance (G, I_0', k). (ii) By Lemma 6.3.7(i) and (ii), $|I_M \cap I| \leq 1$ and

$|I_M \cap S| \leq 2$. As $I_0 \subseteq I$ and $|N(v) \cap (I_M \cup I_0)| > k+3$, there are at least $k+1$ edges between v and I. Since $|E_-| \leq k$, we must have $v \in C$. Moreover, since C is a clique, $C \subseteq N[v]$, and every vertex nonadjacent to v has to be in I. This justifies the marking of $V(G) \setminus N[v]$. Clearly, $(C \setminus \{v\}) \uplus I \uplus S$ is a valid partition of $(G - v, I'_0, k)$. □

Rule 6.3.4. *Let (G, I_0, k) be an annotated instance. Remove all the edges in $E(I_0, I_0)$, and decrease k accordingly.*

Lemma 6.3.11. *Rule 6.3.4 is safe.*

Proof. By the definition of the annotated instance, any solution E_- of (G, I_0, k) contains all the edges in $E(I_0, I_0)$. Moreover, we have that $E_- \setminus E(I_0, I_0)$ is a solution to $G - E(I_0, I_0)$, and its size is at most $k - |E(I_0, I_0)|$. On the other hand, if $(G - E(I_0, I_0), k - |E(I_0, I_0)|)$ is a yes-instance, then any solution of this instance, together with $E(I_0, I_0)$, makes a solution of (G, I_0, k) of size at most k. □

Rule 6.3.5. *Let (G, I_0, k) be an annotated instance where I_0 is an independent set. Introduce p new vertices $v_1, v_2, \ldots, v_p$, where $p = \max_{v \in V(G) \setminus I_0} |N(v) \cap I_0|$. For each vertex $x \in N(I_0)$, make x adjacent to $v_1, \ldots, v_{|N(x) \cap I_0|}$. Remove all vertices in I_0, and mark the set of new vertices.*

The following statement ensures the safeness of Rule 6.3.5. Note that if Rule 6.3.3 is not applicable, then $p \leq k+2$.

Lemma 6.3.12. *Let (G, I_0, k) and (G', I'_0, k) be two annotated instances where $G - I_0 = G' - I'_0$ and both I_0 and I'_0 are independent sets. If $|N_G(x) \cap I_0| = |N_{G'}(x) \cap I'_0|$ for every $x \in V(G) \setminus I_0$, then (G, I_0, k) is a yes-instance if and only if (G', I'_0, k) is a yes-instance.*

Proof. We show that $C \uplus I \uplus S$ is a valid partition of (G, I_0, k) if and only if $C \uplus ((I \setminus I_0) \cup I'_0) \uplus S$ is a valid partition of (G', I'_0, k). Note that

$$|E((I \cup S) \setminus I_0, I_0)| = \sum_{x \in (I \cup S) \setminus I_0} |N_G(x) \cap I_0| = \sum_{x \in (I \cup S) \setminus I'_0} |N_{G'}(x) \cap I'_0|$$
$$= |E((I \cup S) \setminus I'_0, I'_0)|.$$

Since $G - I_0 = G' - I'_0$, and there is no edge in $G[I_0]$ or $G'[I'_0]$, we conclude the proof. □

Rule 6.3.6. *Let (G, I_0, k) be an annotated instance where I_0 is an independent set, and let v be a vertex in C_M. If v is not contained in any $2K_2$ or any I_0-centered P_3, and every C_4 and C_5 that contains v intersects I_0, then remove v from G.*

Lemma 6.3.13. *Rule 6.3.6 is safe.*

Proof. We show that (G, I_0, k) is a yes-instance if and only if $(G - v, I_0, k)$ is a yes-instance, by establishing a sequence of equivalent instances. For each edge xy with $x \in I_0$ and $y \in V(G) \setminus I_0$, introduce a new vertex v_{xy} and make it adjacent to y. Remove all vertices in I_0, and let I'_0 denote the set of new vertices. Let (G', I'_0, k) denote the resulting instance. The equivalence between (G', I'_0, k) and (G, I_0, k) follows from Lemma 6.3.12. Then let (G'', k) denote the graph obtained by applying Rule 6.3.1 to (G', I'_0, k), with K being the added clique.

We argue that v is not contained in any $2K_2$, C_4 or C_5 of G''. Suppose for contradiction that there is a set $F \subseteq V(G'')$ that contains v and induces a $2K_2$, C_4, or C_5 in G''. Since neither the transformation from G to G' nor the transformation from G' to G'' makes any change to $V(G) \setminus I_0$, this set induces the same subgraph in G and G''. Thus, $F \not\subseteq V(G) \setminus I_0$. Moreover, since every vertex in K is universal in $G'' - I_0'$, it follows that $F \cap I_0'$ is not empty. Note that every vertex in I_0' has degree one in G'', we can conclude that $G''[F]$ must be $2K_2$ and $F \cap K = \emptyset$. But then v is contained in either a $2K_2$ or an I_0-centered P_3 in G, a contradiction.

It then follows from Lemma 6.3.2 that (G'', k) is equivalent to $(G'' - v, k)$. To see the equivalence between $(G'' - v, k)$ and $(G - v, I_0, k)$, we apply the reversed operations from G to G''. We first use Rule 6.3.3, applied to $(G'' - v, \emptyset, k)$, to mark all vertices in I_0', then use Lemma 6.3.12 to replace I_0' by I_0, and finally remove vertices in K. The resulting graph is precisely $G - v$. We can thus conclude the proof. □

We call an annotated instance *reduced* if none of Rules 6.3.2–6.3.6 is applicable to this instance.

Lemma 6.3.14. *If a reduced instance* (G, I_0, k) *is a yes-instance, then* $|C_M| \leq 3k\sqrt{2k} + 2k + 2$ *and* $|I_M| \leq k + 3$.

Proof. Let E_- be any solution to (G, I_0, k) with at most k edges and $C \uplus I \uplus S$ a pseudo-split partition pf $G - E_-$. Since Rule 6.3.6 is not applicable, every vertex in C_M is contained in some $2K_2$ or I_0-centered P_3, or some C_4 or C_5 in $G - I_0$. If $\text{opt}(G) = \text{sed}(G)$, then every minimal forbidden structure contains an edge in E_-. If $\text{opt}(G) < \text{sed}(G)$, then all minimal forbidden structures except for one C_5 (it is S) contain an edge in E_-, then $|S \cap C_M| \leq 2$ (by Proposition 6.3.1). Therefore, to bound $|C_M|$, it suffices to count how many vertices in C_M can form a $2K_2$ or I_0-centered P_3, or a C_4 or C_5 in $G - I_0$ with an edge $xy \in E_-$.

- If a vertex $v \in C_M$ is in a $2K_2$ with edge xy, then either $v \in \{x, y\}$ or v is adjacent to neither x nor y. In the first case, no other vertex in C_M can occur in any $2K_2$ with xy. Since $xy \in E(G)$, at least one of them is not in I_0 (Rule 6.3.4). This vertex has at most $\sqrt{2k} + 1$ non-neighbors in C_M. Therefore, the total number of vertices in C_M that can occur in any $2K_2$ with xy is at most $\sqrt{2k} + 1$.

- If xy is an edge in any I_0-centered P_3, then precisely one of them is in I_0. Assume without loss of generality $x \in I_0$. If a vertex $v \in C_M$ is in an I_0-centered P_3 with the edge xy, then either $v = y$, or v is not adjacent to y. If $v = y$, then there is no other vertex in C_M that can be in an I_0-centered P_3 with xy. Since $y \notin I_0$, it has at most $\sqrt{2k} + 1$ non-neighbors in C_M. Thus, the total number of vertices in C_M that can occur in any I_0-centered P_3 containing xy is at most $\sqrt{2k} + 1$.

- If a vertex $v \in C_M$ is in a C_4 or C_5 that contains xy, then v is adjacent to at most one of x and y. Since this C_4 or C_5 is in $G - I_0$, then each of x and y has at most $\sqrt{2k} + 1$ non-neighbors in C_M. Thus, the total number of vertices in C_M that can occur in such a C_4 or C_5 is at most $2\sqrt{2k} + 2$.

Noting that an edge cannot satisfy both the second ($|\{x, y\} \cap I_0| = 1$) and third ($|\{x, y\} \cap I_0| = 0$) categories, we can conclude $|C_M| \leq k(\sqrt{2k} + 2\sqrt{2k} + 2) + 2 = 3k\sqrt{2k} + 2k + 2$.

Since Rule 6.3.2 is not applicable, no vertex in I_M is simplicial. Let E_- be a set of at most k edges such that $G - E_-$ is a pseudo-split graph and $C \uplus I \uplus S$ is a pseudo-split partition of $G - E_-$. Since C is a clique, for each vertex $v \in I_M \cap I$, at least one neighbor of v is in $I \cup S$. Therefore, each vertex $v \in I_M \cap I$ is incident to an edge in the solution E_-. Noting that I_M is an independent set, we have $k \geq |I_M \cap I| \geq |I_M| - 1$, where the second inequality follows from Lemma 6.3.7(i).

Note that $|I_M \cap S| \le 2$ by Lemma 6.3.7(ii). Thus, $|I_M| \le k+3$, and this concludes this proof. □

We use the algorithm described in Figure 6.5. The analysis of the kernels is the same as that of Theorem 6.2.1.

Theorem 6.3.15. *There is an $O(k^{1.5})$-vertex kernel for the pseudo-split edge deletion problem. There is a kernel of $O(k^{1.5})$ vertices and $O(k^{2.5})$ edges for the pseudo-split completion problem.*

Chapter 7

Conclusions and Future Research

In this chapter, I conclude my thesis and provide an overview of topics I plan to pursue in the future. A summary of all the results in this thesis was listed in Section 1.4. My future work will focus on developing kernelization algorithms for graph modification problems, not only for graph classes characterized by finite forbidden induced subgraphs (such as claw-free graphs and line graphs) but also for graph classes with infinite obstructions (such as chordal graphs and interval graphs). Additionally, I plan to investigate characterizations of circular-arc graphs and their subclasses.

7.1 Conclusions

In this thesis, we investigate edge modification problems for several graph classes that are characterized by finite forbidden induced subgraphs, and we provide a polynomial kernel for each of these problems.

In Chapter 3, we study the cluster (P_3-free) edge deletion problem and provide a $2k$-vertex kernel for it. Additionally, we use the same kernelization algorithm to the strong triadic closure problem and obtain a $2k$-vertex kernel.

In Chapter 4, we present a kernel with $38k$ vertices for the paw-free completion problem. We believe the size of the kernel can be improved to $16k$ by careful analysis, but we leave an open problem of whether a kernel with a size smaller than $10k$ can be obtained.

In Chapter 5, we show a very simple kernelization algorithm for the trivially perfect ($\{P_4, C_4\}$-free) completion problem. The resulting kernel contains at most $2k^2 + 2k$ vertices.

In Chapter 6, we study the kernelization algorithms for the split edge deletion (completion) problem and the pseudo-split edge deletion (completion) problem, and show a kernel with $O(k^{1.5})$ vertices for both problems.

7.2 Future Research

For decades, researchers have been intrigued by the question of which graph modification problems admit polynomial kernels. I am also eager to contribute my efforts to this area of research.

7.2.1 Polynomial kernels for edge modification problems

For graph classes that are characterized by only one simple forbidden induced subgraph, denoted as H-free graphs. For these graphs, the question of whether there exists a polynomial kernel for the corresponding edge modification problems has been studied extensively. Specifically, it has been shown that for all graphs on three vertices and for all graphs on four vertices except for the claw (a star on four vertices), a polynomial kernel exists. We use S_ℓ to denote a star with ℓ vertices. For the S_3-free (cluster) edge deletion problem, we have shown the existence of a $2k$-vertices kernel. However, for the S_{11}-free edge deletion problem, Cai [25] proved that a polynomial kernel does not exist unless NP $\subseteq$ coNP/ poly. The question of whether the S_ℓ-free edge deletion problem with $4 \leq \ell \leq 10$ has a polynomial kernel remains open. Solving the following open problem is an important step to closing the gap [41, 24, 53, 43, 39].

Open problem 7.2.1. *Does the claw-free edge deletion problem admit a polynomial kernel?*

We are also interested in this problem, as well as the claw-free completion problem and the claw-free editing problem.

Open problem 7.2.2. *Does the claw-free completion problem admit a polynomial kernel?*

Open problem 7.2.3. *Does the claw-free editing problem admit a polynomial kernel?*

To solve the problem of finding a polynomial kernel for the claw-free edge deletion problem, we could explore graph classes related to the claw-free graphs. For instance, we can examine subclasses of claw-free graphs, such as line graphs and proper interval graphs.

A line graph is a graph that does not contain any of the nine graphs shown in Figure 7.1 as an induced subgraph [4]. Recently, Eiben and Lochet [58] showed that the edge deletion problem toward line graphs admits a polynomial kernel with $O(k^5)$ vertices. However, the existence of polynomial kernels for the completion problem and the editing problem toward line graphs is still unknown, and it is included in my lists of future work.

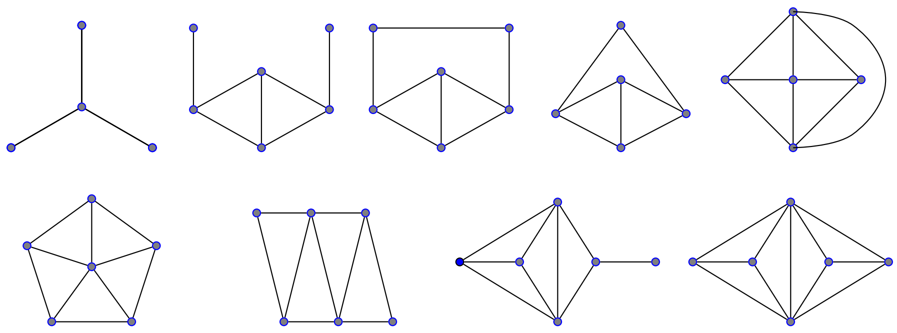


Figure 7.1: The nine minimal forbidden induced subgraphs of line graphs.

Open problem 7.2.4. *Does the line graph completion problem admit a polynomial kernel?*

Open problem 7.2.5. *Does the line graph editing problem admit a polynomial kernel?*

The proper interval graphs is another subclass of claw-free graphs, which is characterized by an infinite number of minimal forbidden induced subgraphs. A graph is a proper interval if it contains no claw, net, tent, or hole (cycles of length larger than three) as an induced subgraph [158]. Proper interval graphs is also known as claw-free interval graphs [137, 16]. The proper interval completion problem has been shown to admit a polynomial kernel with $O(k^3)$ vertices [10], while the proper interval edge deletion problem and the proper interval editing problem have not been shown to have such polynomial kernels yet.

Open problem 7.2.6. *Does the proper interval edge deletion problem admit a polynomial kernel?*

Open problem 7.2.7. *Does the proper interval editing problem admit a polynomial kernel?*

Chordal graphs and interval graphs are two important graph classes characterized by an infinite number of minimal forbidden induced subgraphs. The chordal completion problem has been shown to have a kernel with $O(k^2)$ vertices [123]. However, for the edge deletion and editing problems toward chordal graphs, no polynomial kernel has yet been discovered.

I am interested in the interval completion problem. While it has been proven to be FPT in [156, 26], we still do not know if a polynomial kernel exists for it. It is known that interval graphs is a subclass of chordal graphs and a superclass of proper interval graphs. Interestingly, the completion problem towards proper interval graphs and chordal graphs both have polynomial kernels. This situation makes the question of whether a polynomial kernel exists for the interval completion problem even more intriguing. To approach this problem, I plan to first consider a simpler problem: the completion problem from a chordal graph to an interval graph. This problem has already been proven to be NP-complete in [131]. My primary objective is to explore the existence of a polynomial kernel for this problem.

CHORDAL → INTERVAL COMPLETION

Input: A chordal graph G and a nonnegative integer k.

Output: Is there a set E_+ of at most k edges such that $G+E_+$ is an interval graph?

Open problem 7.2.8. *Does* CHORDAL → INTERVAL COMPLETION *admit a polynomial kernel?*

For this problem, we have some observations and some ideas that have not yet been proven. We know that a graph is a chordal graph if and only if it has a clique tree [11], and a graph is an interval graph if and only if it has a clique path [72, 76]. Starting from a clique tree T of a chordal graph G with a minimum number of leaves, we observe that the number of leaves can be reduced at most by one by adding an edge to G. Thus, the number of leaves of T is at most $k + 2$, and the number of nodes in T of degree larger than two is at most $k + 1$. If we can bound the number of nodes with degree two in T and the number of vertices contained in every node of T in the polynomial of k, then we can obtain a polynomial kernel.

We hope this work could give us some hints for kernelization algorithms for the interval completion problem.

Open problem 7.2.9. *Does the interval completion problem admit a polynomial kernel?*

For the open problems we discussed above, we could try to use the modulator method first. Regarding the edge modification problems for claw-free graphs and line graphs, we cloud obtain a modulator by finding a maximal packing of vertex-disjoin claws and the induced subgraphs illustrated in Figure 7.1. Since there are infinite number of forbidden induced subgraphs for (proper) interval graphs, we cannot obtain a modulator by finding a maximal packing of their forbidden induced subgraphs. Therefore, we need to explore alternative methods to obtain the modulator, such as utilizing approximation algorithms with a constant ratio. However, the best-known result is the $O(\log n)$ approximation algorithm for the interval completion problem [134]. Hence, it is challenging to employ the modulator methods for the kernelization algorithms to the interval completion problem.

7.2.2 Characterizations of normal circular-arc graphs

Characterizing circular-arc graphs and its subclasses by forbidden induced subgraphs is also one of my main areas of focus for future work. A circular-arc graph is an intersection graph of arcs on a circle and is a generalization of an interval graph. Lekkerkerker and Boland [109] showed the set of forbidden induced subgraphs for interval graphs in the 1960s, the same is not true for circular-arc graphs, whose set of forbidden induced subgraphs remains unkonwn after years of research [153, 154, 61, 111, 17, 70]. On the positive side, the forbidden subgraph characterizations of several subclasses of circular-arc graphs are known to us, such as proper circular-arc graphs (in which no arc is properly contained in any other arc) [154], unit circular-arc graphs (in which all arcs have the same length) [154], and normal Helly circular-arc graphs (in which no two arcs cover the circle and every family of pairwise intersecting arcs shares a common point) [29]. Trotter and Moore [151] characterized two-clique circular arc graphs in terms of forbidden subgraphs. Lin and Szwarcfiter [111] had given the minimal forbidden induced subgraphs of Helly circular-arc graphs (in which every family of pairwise intersecting arcs shares a common point) when the input graph is a circular-arc graph.

Normal circular-arc graphs (in which no two arcs cover the circle) is a subclass of circular-arc graphs for which we do not yet know the forbidden induced subgraph characterization. I want to study the characterization of normal circular-arc graphs by minimal forbidden induced subgraphs when the input graph is a circular-arc graph.

Hell and Huang [92] showed that if G is a co-bipartite circular-arc graph, then G is a normal circular-arc graph if and only if $\overline{G}$ is an interval bigraph. They also provided the forbidden induced subgraph characterization of interval bigraphs. Here we need to consider that G is a circular-arc graph but not co-bipartite, which means that the vertices of G cannot be covered by two cliques.

For a minimal circular-arc graph H that is not normal, there are two arcs that cover the circle in any arc model of H. We consider whether these two arcs are fixed in any arc model. There are three cases to consider: (*i*) both of the two arcs are fixed in any arc model, (*ii*) only one of them is fixed in any arc model, (*iii*) none of them is fixed.

Next, we give some definitions. A circular-arc model is an ordered pair $(C, \mathcal{A})$, where C is a circle, and $\mathcal{A}$ is a family of arcs of C. Unless explicitly stated, we always traverse C in the clockwise direction for traverses C. For a vertex v in a circular-arc graph, we write $A(v) = [\mathtt{lp}(v), \mathtt{rp}(v)]$ to denote the arc of v that traversing from $\mathtt{lp}(v)$ to $\mathtt{rp}(v)$, where $\mathtt{lp}(v)$ ($\mathtt{rp}(v)$) is the beginning point (ending point) of $A(v)$. For a circular-arc graph G, we consider the arc model of G. Let C be a circle and let p and q be the two endpoints of a diameter of C. We use P (respectively Q) to denote the open segment from p to q (respectively from q to p), where $P = [p, q]$.

We begin by examining the minimal non-normal circular-arc graphs that have two fixed arcs covering the circle in any arc model. Let H be a circular-arc graph, and consider every arc model of H that contains eight arcs satisfying the following conditions:

(i) Four arcs $A(a), A(b), A(c)$, and $A(d)$ that do not intersect each other. Two of these arcs, $A(a)$ and $A(b)$, are in P, while the other two, $A(c)$ and $A(d)$, are in Q,

(ii) Two arcs $A(u)$ and $A(v)$ that do not intersect each other. Arc $A(u)$ is located between $\texttt{lp}(a)$ and $\texttt{rp}(b)$, while arc $A(v)$ is located between $\texttt{lp}(c)$ and $\texttt{rp}(d)$.

(iii) Two arcs, $A(x)$ and $A(y)$, such that $A(x)$ intersects all seven arcs but $A(u)$ and $A(y)$ intersects all seven arcs but $A(v)$.

If these conditions hold, we can conclude that H is not a normal circular-arc graph, and arcs $A(x)$ and $A(y)$ cover the circle (refer to Figure 7.2).

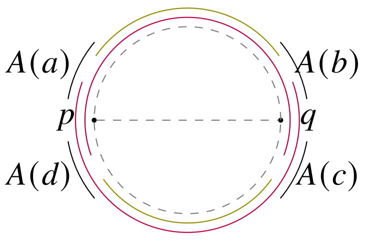


Figure 7.2: Two arcs A_x and A_y cover the circle. The two olive arcs are A_u and A_v, the two purple arcs are A_x and A_y.

We have discovered many such minimal non-normal circular-arc graphs, and we believe that there is a finite number of such graphs. Figure 7.3 showcases three of these graphs that we have discovered.

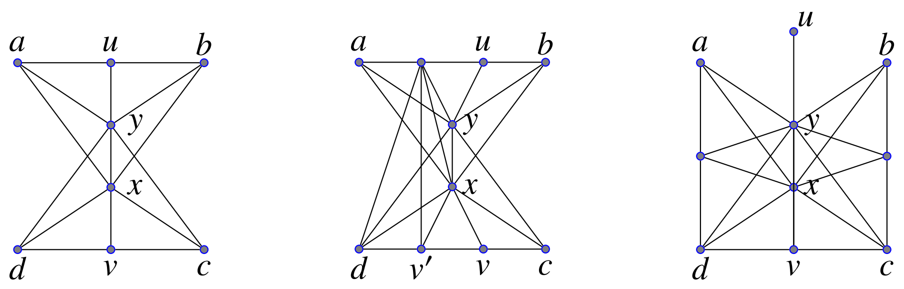


Figure 7.3: Three minimal non-normal circular-arc graphs with two vertices x and y such that the corresponding arcs cover the circle in any arc model.

Now we consider the minimal non-normal circular-arc graphs that have only one fixed arc covering the circle with the other one arc in any arc model. Upon observation, we have noticed that these graphs contain a universal vertex, and we suspect that its corresponding arc is the fixed arc that covers the circle, although we have not yet established proof for this.

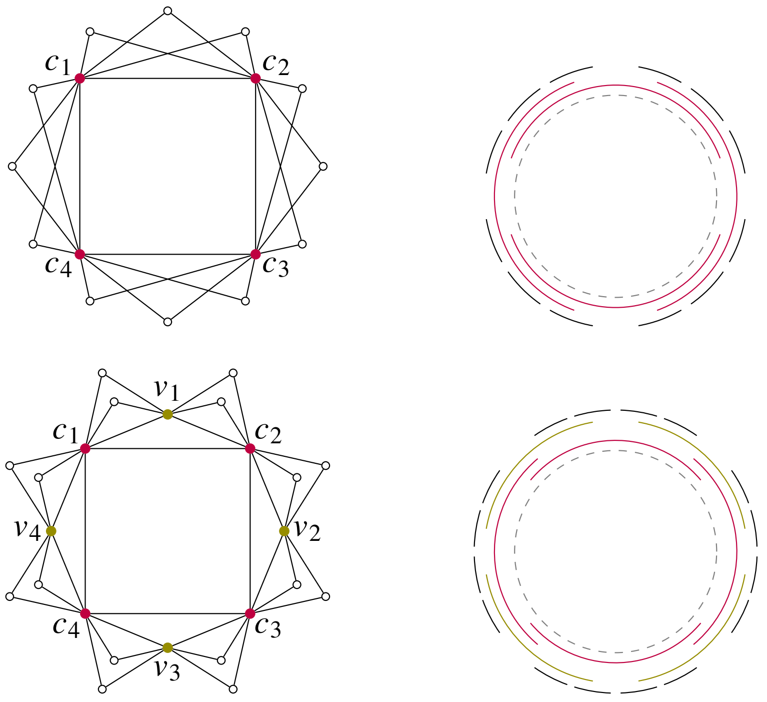


Figure 7.4: Tow hole-stars (removing the universal vertices).

We define a *hole-star* as a graph H that contains a universal vertex and an induced cycle $C_\ell = \{c_1, \ldots, c_\ell\}$, where $\ell \geq 3$, such that for any pair of consecutive vertices c_i and c_{i+1} on C_ℓ, there are either three pairwise nonadjacent vertices that are only adjacent to c_i and c_{i+1}, or there is a vertex v_i only adjacent to c_i and c_{i+1} and two pairwise nonadjacent vertices that are only adjacent to c_i and v_i. Figure 7.4 illustrates two examples of hole-star graphs. It is easy to see that the arc of the universal vertex and an arc of a vertex on the C_4 (or an arc of a vertex v_i) cover the circle. In this case, there are infinitely many such minimal non-normal circular-arc graphs.

To study this case more closely, we aim to determine if the number of such minimal non-normal circular-arc graphs is finite in chordal graphs. Specifically, we consider a chordal circular-arc graph G which contains a universal vertex.

Suppose that G is a minimal non-normal circular-arc graph. As G is minimal, it has exactly one universal vertex, which we assume is u. Since G is a minimal non-normal circular-arc graph, the subgraph $G - u$ is a normal circular-arc graph. Let $\mathcal{M}$ be an arc model of $G-u$. Because $G-u$ is chordal, the number of arcs covering the circle is at most three. Since $G-u$ is a normal circular-arc graph, the number of arcs cover the circle is exactly three. Let $A(v_1)$, $A(v_2)$, and $A(v_3)$ cover the circle, where v_1, v_2, and v_3 are vertices in $G - u$. The six endpoints of these three arcs divide the circle into six sections. Let $X_1 = A(v_1) \cap A(v_2)$, $X_2 = A(v_2) \cap A(v_3)$, $X_3 = A(v_3) \cap A(v_1)$, $Y_1 = A(v_1) \setminus (A(v_2) \cup A(v_3))$, $Y_2 = A(v_2) \setminus (A(v_1) \cup A(v_3))$, and $Y_3 = A(v_3) \setminus (A(v_1) \cup A(v_2))$. See Figure 7.5 for an illustration.

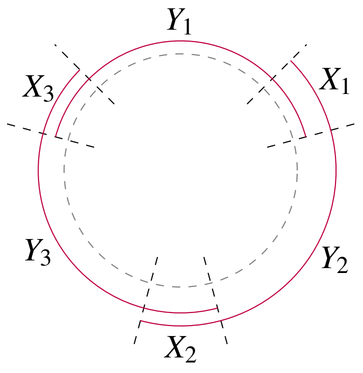


Figure 7.5: An arc model for $G - u$.

Since G is a minimal non-normal circular-arc graph, we can construct a non-normal arc model for G by adding the arc $A(u)$ to $\mathcal{M}$, let $\mathcal{M}'$ to denote it. The circle of $\mathcal{M}'$ is covered by $A(u)$ and another arc in $\mathcal{M}$. Every arc in $\mathcal{M}$ is either properly contained in one of $A(v_1)$, $A(v_2)$, and $A(v_3)$, or properly contains one of X_1, X_2, and X_3.

We can assume that there is no arc properly containing $A(v_1)$, $A(v_2)$, or $A(v_3)$ since we can replace one of $A(v_1)$, $A(v_2)$, and $A(v_3)$ with the arc otherwise. Then we can assume that the arc that covers the circle with $A(u)$ is either one of $A(v_1)$, $A(v_2)$, and $A(v_3)$, or an arc properly contains one of X_1, X_2, and X_3. If $A(u)$ covers the circle with one of $A(v_1)$, $A(v_2)$, and $A(v_3)$, then $A(u)$ intersects all of X_1, X_2, and X_3. If $A(u)$ covers the circle with an arc that properly contains one of X_1, X_2, and X_3, then $A(u)$ intersects at least two of the three segments.

If $A(u)$ intersects X_1 in any arc model of G, there is an arc $A(w)$ that is properly contained in X_1. In this case, there is an arc at each end of X_1 that intersects X_1 and does not intersect $A(w)$ (see an example in Figure 7.6(a)).

Now suppose that $A(u)$ does not intersect X_1 in an arc model of G. Then $A(u)$ must cover the circle with an arc $A(x)$ that properly contains X_1 in the model. Then we can assume that there are no arcs that are properly contained in X_1. Since $A(u)$ and $A(x)$ cover the circle, there is at least one arc that is properly contained in $A(x) \cap A(v_1)$ and one arc that is properly contained in $A(x) \cap A(v_2)$. Suppose $A(u_1)$ is properly contained in $A(x) \cap A(v_1)$ and $A(u_2)$ is properly contained in $A(x) \cap A(v_2)$. By assumption, we can get that both $A(u_1)$ and $A(u_2)$ are not properly contained in X_1. By considering the intersection of $A(u_1)$ and $A(u_2)$, we obtain the graphs in Figure 7.6(b–e). Considering the intersection of X_2 and $A(u)$ and the intersection of X_3 and $A(u)$, we can combine these five cases to obtain 35 such non-normal circular-arc graphs. It is easy to see that we can obtain a minimal non-normal circular-arc graph by replacing the triangle $v_1v_2v_3$ with a hole.

For minimal non-normal circular-arc graphs where none of the covering arcs is

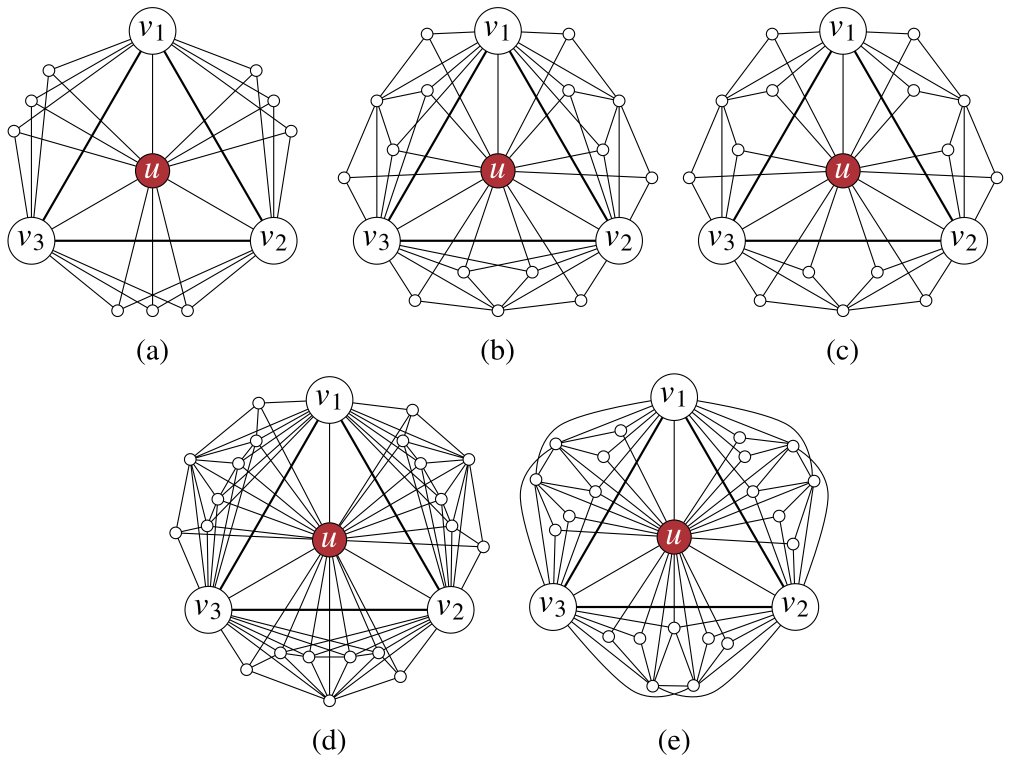
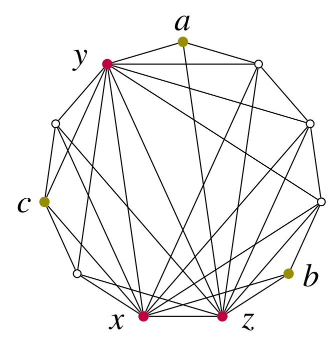
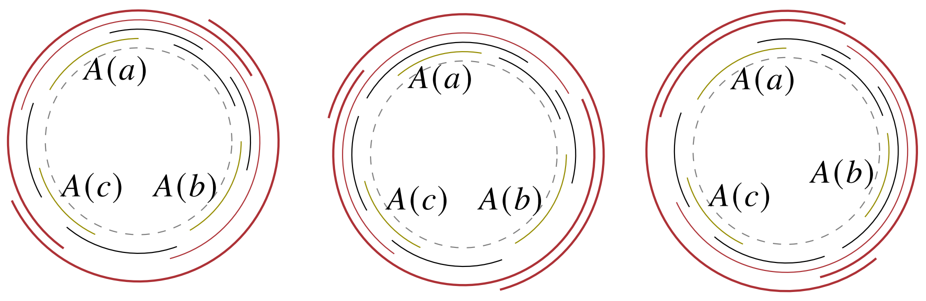


Figure 7.6: Five minimal non-normal circular-arc graphs.

fixed, we have only found a few examples of such graphs. As shown in Figure 7.7, the vertices x, y, and z are adjacent to all vertices but a, b, and c, respectively. In different arc models of the graph (as depicted Figure 7.7(b–d)), the pairs of arcs that cover the circle are different.



(a)



(b) (c) (d)

Figure 7.7: A minimal forbidden subgraph. In (b), the arcs of y and x cover the circle. In (c), the arcs of x and z cover the circle. In (d), the arcs of y and z cover the circle.

References

[1] Tülay Adalı and Antonio Ortega. “Applications of graph theory [Scanning the Issue]”. In: *Proceedings of the IEEE* 106.5 [2018], pp. 784–786. DOI: 10.1109/JPROC.2018.2820300.

[2] Akanksha Agrawal, Pranabendu Misra, Saket Saurabh, and Meirav Zehavi. “Interval vertex deletion admits a polynomial kernel”. In: *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM. 2019, pp. 1711–1730. DOI: 10.1137/1.9781611975482.103.

[3] R. Balasubramanian, Michael R. Fellows, and Venkatesh Raman. “An improved fixed-parameter algorithm for vertex cover”. In: *Information Processing Letters* 65.3 [1998], pp. 163–168. DOI: 10.1016/S0020-0190(97)00213-5.

[4] Lowell W. Beineke. “Characterizations of derived graphs”. In: *Journal of Combinatorial theory* 9.2 [1970], pp. 129–135. DOI: 10.1016/S0021-9800(70)80019-9.

[5] Richard Bellman. “Dynamic programming”. In: *Science* 153.3731 [1966], pp. 34–37. DOI: 10.1126/science.153.3731.34.

[6] Richard Bellman. “Dynamic programming treatment of the travelling salesman problem”. In: *Journal of the ACM (JACM)* 9.1 [1962], pp. 61–63. DOI: 10.1145/321105.321111.

[7] Richard Bellman. “The theory of dynamic programming”. In: *Bulletin of the American Mathematical Society* 60.6 [1954], pp. 503–515. DOI: 10.1090/S0002-9904-1954-09848-8.

[8] Stéphane Bessy, Marin Bougeret, Dimitrios M. Thilikos, and Sebastian Wiederrecht. “Kernelization for Graph Packing Problems via Rainbow Matching”. In: *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM. 2023, pp. 3654–3663. DOI: 10.1137/1.9781611977554.ch139.

[9] Stéphane Bessy, Fedor V. Fomin, Serge Gaspers, Christophe Paul, Anthony Perez, Saket Saurabh, and Stéphan Thomassé. “Kernels for feedback arc set in tournaments”. In: *Journal of Computer and System Sciences* 77.6 [2011], pp. 1071–1078. DOI: 10.1016/j.jcss.2010.10.001.

[10] Stéphane Bessy and Anthony Perez. “Polynomial kernels for proper interval completion and related problems”. In: *Information and Computation* 231 [2013], pp. 89–108. DOI: 10.1016/j.ic.2013.08.006.

[11] Jean R. S. Blair and Barry Peyton. “An introduction to chordal graphs and clique trees”. In: *Graph theory and sparse matrix computation*. Springer. 1993, pp. 1–29. DOI: 10.1007/978-1-4613-8369-7_1.

[12] Ivan Bliznets, Marek Cygan, Pawel Komosa, Lukáš Mach, and Michał Pilipczuk. “Lower bounds for the parameterized complexity of minimum fill-in and other completion problems”. In: *Proceedings of the Twenty-*

Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). SIAM. 2016, pp. 1132–1151. DOI: 10.1145/3381426.

[13] Sebastian Böcker, Sebastian Briesemeister, Quang Bao Anh Bui, and Anke Truß. "Going weighted: Parameterized algorithms for cluster editing". In: *Theoretical Computer Science* 410.52 [2009], pp. 5467–5480. DOI: 10.1016/j.tcs.2009.05.006.

[14] Sebastian Böcker and Peter Damaschke. "Even faster parameterized cluster deletion and cluster editing". In: *Information Processing Letters* 111.14 [2011], pp. 717–721. DOI: 10.1016/j.ipl.2011.05.003.

[15] Hans L. Bodlaender, Leizhen Cai, Jianer Chen, Michael R. Fellows, Jan Arne Telle, and Dániel Marx. "Open problems in parameterized and exact computation-IWPEC 2006". In: *UU-CS* 2006 [2006].

[16] Kenneth P. Bogart and Douglas B. West. "A Short Proof that 'Proper= Unit'". In: *Discrete Mathematics* 201 [1999], pp. 21–23.

[17] Flavia Bonomo, Guillermo Durán, Luciano N. Grippo, and Martín Darío Safe. "Partial characterizations of circular-arc graphs". In: *Journal of Graph Theory* 61.4 [2009], pp. 289–306. DOI: 10.1002/jgt.20379.

[18] Kellogg S. Booth and George S. Lueker. "Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms". In: *Journal of computer and system sciences* 13.3 [1976], pp. 335–379. DOI: 10.1016/S0022-0000(76)80045-1.

[19] Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph classes: a survey*. SIAM Monographs on Discrete Mathematics and Applications, 1999.

[20] Pablo Burzyn, Flavia Bonomo, and Guillermo Durán. "NP-completeness results for edge modification problems". In: *Discrete Applied Mathematics* 154.13 [2006], pp. 1824–1844. DOI: 10.1016/j.dam.2006.03.031.

[21] Jonathan F Buss and Judy Goldsmith. "Nondeterminism within P^*". In: *SIAM Journal on Computing* 22.3 [1993], pp. 560–572. DOI: 10.1137/0222038.

[22] Guo-Ray Cai and Yu-Geng Sun. "The minimum augmentation of any graph to a K-edge-connected graph". In: *Networks* 19.1 [1989], pp. 151–172. DOI: 10.1002/net.3230190112.

[23] Leizhen Cai. "Fixed-parameter tractability of graph modification problems for hereditary properties". In: *Information Processing Letters* 58.4 [1996], pp. 171–176. DOI: 10.1016/0020-0190(96)00050-6.

[24] Leizhen Cai and Yufei Cai. "Incompressibility of H-free edge modification problems". In: *Algorithmica* 71.3 [2015], pp. 731–757. DOI: 10.1007/s00453-014-9937-x.

[25] Yufei Cai. "Polynomial kernelisation of H-free edge modification problems". PhD thesis. Chinese University of Hong Kong, 2012.

[26] Yixin Cao. "Linear recognition of almost interval graphs". In: *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM. 2016, pp. 1096–1115. DOI: 10.1137/1.9781611974331.ch77.

[27] Yixin Cao. "Recognizing (Unit) Interval Graphs by Zigzag Graph Searches". In: *Symposium on Simplicity in Algorithms (SOSA)*. SIAM. 2021, pp. 92–106. DOI: 10.1137/1.9781611976496.11.

[28] Yixin Cao and Jianer Chen. “Cluster editing: Kernelization based on edge cuts”. In: *Algorithmica* 64.1 [2012], pp. 152–169. DOI: 10.1007/s00453-011-9595-1.

[29] Yixin Cao, Luciano N. Grippo, and Martín D. Safe. “Forbidden induced subgraphs of normal Helly circular-arc graphs: Characterization and detection”. In: *Discrete Applied Mathematics* 216 [2017], pp. 67–83. DOI: 10.1016/j.dam.2015.08.023.

[30] Yixin Cao, Yuping Ke, Yota Otachi, and Jie You. “Vertex deletion problems on chordal graphs”. In: *Theoretical Computer Science* 745 [2018], pp. 75–86. DOI: 10.1016/j.tcs.2018.05.039.

[31] Yixin Cao, Ashutosh Rai, R.B. Sandeep, and Junjie Ye. “A polynomial kernel for diamond-free editing”. In: *Algorithmica* 84.1 [2022], pp. 197–215. DOI: 10.1007/s00453-021-00891-y.

[32] Jianer Chen, Iyad A. Kanj, and Weijia Jia. “Vertex cover: further observations and further improvements”. In: *Journal of Algorithms* 41.2 [2001], pp. 280–301. DOI: 10.1006/jagm.2001.1186.

[33] Jianer Chen and Jie Meng. “A 2k kernel for the cluster editing problem”. In: *Journal of Computer and System Sciences* 78.1 [2012], pp. 211–220. DOI: 10.1016/j.jcss.2011.04.001.

[34] Hyeong-Ah Choi, Kazuo Nakajima, and Chong S. Rim. “Graph bipartization and via minimization”. In: *SIAM Journal on Discrete Mathematics* 2.1 [1989], pp. 38–47. DOI: 10.1137/0402004.

[35] Benny Chor, Mike Fellows, and David Juedes. “Linear kernels in linear time, or how to save k colors in $O(n^2)$ steps”. In: *Graph-Theoretic Concepts in Computer Science: 30th International Workshop, WG 2004, Bad Honnef, Germany, June 21-23, 2004. Revised Papers 30*. Springer. 2004, pp. 257–269. DOI: 10.1007/978-3-540-30559-0_22.

[36] Vclav Chvtal and Peter L. Hammer. “Aggregation of inequalities in integer programming”. In: *Ann. Discrete Math* 1 [1977], pp. 145–162.

[37] Vclav Chvtal and Peter L. Hammer. *Set-packing Problems And Threshold Graphs, CORR 73-21*. 1973.

[38] Derek G. Corneil, Stephan Olariu, and Lorna Stewart. “The LBFS structure and recognition of interval graphs”. In: *SIAM Journal on Discrete Mathematics* 23.4 [2010], pp. 1905–1953. DOI: 10.1137/S0895480100373455.

[39] Christophe Crespelle, Pål Grønås Drange, Fedor V. Fomin, and Petr A. Golovach. “A survey of parameterized algorithms and the complexity of edge modification”. In: *arXiv preprint arXiv:2001.06867* [2020]. DOI: 10.48550/arXiv.2001.06867.

[40] Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*. Vol. 5. 4. Springer, 2015.

[41] Marek Cygan, Lukasz Kowalik, and Marcin Pilipczuk. “Open problems from workshop on kernels”. In: *Retrieved on September* 3 [2013], p. 2018.

[42] Marek Cygan, Dániel Marx, Marcin Pilipczuk, and Michał Pilipczuk. “Hitting forbidden subgraphs in graphs of bounded treewidth”. In: *Information*

and Computation 256 [2017], pp. 62–82. DOI: 10.1016/j.ic.2017.04.009.

[43] Marek Cygan, Marcin Pilipczuk, Michał Pilipczuk, Erik Jan Van Leeuwen, and Marcin Wrochna. "Polynomial kernelization for removing induced claws and diamonds". In: *Theory of Computing Systems* 60 [2017], pp. 615–636. DOI: 10.1007/s00224-016-9689-x.

[44] Elias Dahlhaus, David S. Johnson, Christos H. Papadimitriou, Paul D. Seymour, and Mihalis Yannakakis. "The complexity of multiterminal cuts". In: *SIAM Journal on Computing* 23.4 [1994], pp. 864–894. DOI: 10.1137/S0097539792225297.

[45] Peter Damaschke. "Bounded-degree techniques accelerate some parameterized graph algorithms". In: *International Workshop on Parameterized and Exact Computation*. Springer. 2009, pp. 98–109. DOI: 10.1007/978-3-642-11269-0_8.

[46] Peter Damaschke and Olof Mogren. "Editing the simplest graphs". In: *International Workshop on Algorithms and Computation*. Springer. 2014, pp. 249–260. DOI: 10.1007/978-3-319-04657-0_24.

[47] Rodney G. Downey and Michael R. Fellows. "Fixed-parameter intractability". In: *1992 Seventh Annual Structure in Complexity Theory Conference*. IEEE Computer Society. 1992, pp. 36–37. DOI: 10.1109/SCT.1992.215379.

[48] Rodney G. Downey and Michael R. Fellows. "Fixed-parameter tractability and completeness". In: *Complexity Theory: Current Research*. 1992, pp. 191–225.

[49] Rodney G. Downey and Michael R. Fellows. “Fixed-parameter tractability and completeness I: Basic results”. In: *SIAM Journal on computing* 24.4 [1995], pp. 873–921. DOI: 10.1137/S0097539792228228.

[50] Rodney G. Downey and Michael R. Fellows. “Fixed-parameter tractability and completeness II: On completeness for W [1]”. In: *Theoretical Computer Science* 141.1-2 [1995], pp. 109–131. DOI: 10.1016/0304-3975(94)00097-3.

[51] Rodney G. Downey and Michael R. Fellows. *Parameterized complexity*. Springer Science & Business Media, 2012.

[52] Rodney G. Downey, Michael R. Fellows, and Ulrike Stege. “Parameterized complexity: A framework for systematically confronting computational intractability”. In: *Contemporary Trends in Discrete Mathematics* 49 [1997], pp. 49–99.

[53] Pål Grønås Drange. “Parameterized graph modification algorithms”. PhD thesis. The University of Bergen, Norway, 2015.

[54] Pål Grønås Drange, Markus Sortland Dregi, Daniel Lokshtanov, and Blair D. Sullivan. “On the threshold of intractability”. In: *Proceedings of the 23rd Annual European Symposium on Algorithms (ESA)*. Springer, 2015, pp. 411–423. DOI: 10.1007/978-3-662-48350-3_35.

[55] Pål Grønås Drange, Fedor V. Fomin, Michał Pilipczuk, and Yngve Villanger. “Exploring the subexponential complexity of completion problems”. In: *ACM Transactions on Computation Theory (TOCT)* 7.4 [2015], pp. 1–38. DOI: 10.1145/2799640.

[56] Pål Grønås Drange and Michał Pilipczuk. “A polynomial kernel for trivially perfect editing”. In: *Algorithmica* 80.12 [2018], pp. 3481–3524. DOI: 10.1007/s00453-017-0401-6.

[57] Maël Dumas, Anthony Perez, and Ioan Todinca. “A cubic vertex-kernel for trivially perfect editing”. In: *Algorithmica* 85.4 [2023], pp. 1091–1110. DOI: 10.1007/s00453-022-01070-3.

[58] Eduard Eiben, William Lochet, and Saket Saurabh. “A Polynomial Kernel for Paw-Free Editing”. In: *15th International Symposium on Parameterized and Exact Computation (IPEC)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2020.

[59] Paul Erdös and Richard Rado. “Intersection theorems for systems of sets”. In: *Journal of the London Mathematical Society* 1.1 [1960], pp. 85–90. DOI: 10.1112/jlms/s1-35.1.85.

[60] Kapali P. Eswaran and Robert E. Tarjan. “Augmentation problems”. In: *SIAM Journal on Computing* 5.4 [1976], pp. 653–665. DOI: 10.1137/0205044.

[61] Tomás Feder, Pavol Hell, and Jing Huang. “List homomorphisms and circular arc graphs”. In: *Combinatorica* 19.4 [1999], pp. 487–506. DOI: 10.1007/s004939970003.

[62] Michael Fellows, Michael Langston, Frances Rosamond, and Peter Shaw. “Efficient parameterized preprocessing for cluster editing”. In: *International Symposium on Fundamentals of Computation Theory (FCT)*. Springer. 2007, pp. 312–321. DOI: 10.1007/978-3-540-74240-1_27.

[63] Michael R. Fellows, Jiong Guo, Christian Komusiewicz, Rolf Niedermeier, and Johannes Uhlmann. “Graph-based data clustering with overlaps”. In: *Discrete Optimization* 8.1 [2011], pp. 2–17. DOI: 10.1016/j.disopt.2010.09.006.

[64] Stéphane Foldes and Peter L. Hammer. *Split graphs*. Universität Bonn. Institut für Ökonometrie und Operations Research, 1976.

[65] Fedor V. Fomin and Petteri Kaski. “Exact exponential algorithms”. In: *Communications of the ACM* 56.3 [2013], pp. 80–88. DOI: 10.1145/2428556.2428575.

[66] Fedor V. Fomin, Dieter Kratsch, Ioan Todinca, and Yngve Villanger. “Exact algorithms for treewidth and minimum fill-in”. In: *SIAM Journal on Computing* 38.3 [2008], pp. 1058–1079. DOI: 10.1137/050643350.

[67] Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: theory of parameterized preprocessing*. Cambridge University Press, 2019.

[68] Fedor V. Fomin and Saket Saurabh. “Kernelization Methods for Fixed-Parameter Tractability”. In: *Tractability: Practical Approaches to Hard Problems*. Ed. by Lucas Bordeaux, Youssef Hamadi, and PushmeetEditors Kohli. Cambridge University Press, 2014, pp. 260–282. DOI: 10.1017/CBO9781139177801.010.

[69] Fedor V. Fomin and Yngve Villanger. “Subexponential parameterized algorithm for minimum fill-in”. In: *SIAM Journal on Computing* 42.6 [2013], pp. 2197–2216. DOI: 10.1137/11085390X.

[70] Mathew Francis, Pavol Hell, and Juraj Stacho. “Forbidden structure characterization of circular-arc graphs and a certifying recognition algorithm”. In: *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM. 2014, pp. 1708–1727. DOI: 10.1137/1.9781611973730.114.

[71] András Frank. “Augmenting graphs to meet edge-connectivity requirements”. In: *SIAM Journal on Discrete Mathematics* 5.1 [1992], pp. 25–53. DOI: 10.1137/0405003.

[72] Delbert Fulkerson and Oliver Gross. “Incidence matrices and interval graphs”. In: *Pacific Journal of Mathematics* 15.3 [1965], pp. 835–855. DOI: 10.2140/pjm.1965.15.835.

[73] Yong Gao, Donovan R. Hare, and James Nastos. “The cluster deletion problem for cographs”. In: *Discrete Mathematics* 313.23 [2013], pp. 2763–2771. DOI: 10.1016/j.disc.2013.08.017.

[74] Serge Gaspers, Dieter Kratsch, Mathieu Liedloff, and Ioan Todinca. “Exponential time algorithms for the minimum dominating set problem on some graph classes”. In: *ACM Transactions on Algorithms* 6.1 [2009], pp. 1–21. DOI: 10.1145/1644015.1644024.

[75] Esha Ghosh, Sudeshna Kolay, Mrinal Kumar, Pranabendu Misra, Fahad Panolan, Ashutosh Rai, and M.S. Ramanujan. “Faster parameterized algorithms for deletion to split graphs”. In: *Algorithmica* 71.4 [2015], pp. 989–1006. DOI: 10.1007/s00453-013-9837-5.

[76] Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs.* Elsevier, 2004.

[77] Martin Charles Golumbic. “Trivially perfect graphs”. In: *Discrete Mathematics* 24.1 [1978], pp. 105–107. DOI: 10.1016/0012-365X(78)90178-4.

[78] Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. “Automated generation of search tree algorithms for hard graph modification problems”. In: *Algorithmica* 39.4 [2004], pp. 321–347. DOI: 10.1007/s00453-004-1090-5.

[79] Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. “Graph-modeled data clustering: Exact algorithms for clique generation”. In: *Theory of Computing Systems* 38.4 [2005], pp. 373–392. DOI: 10.1007/s00224-004-1178-y.

[80] Martin Grohe. “Logic, graphs, and algorithms.” In: *Logic and automata* 2 [2008], pp. 357–422.

[81] Niels Grüttemeier and Christian Komusiewicz. “On the relation of strong triadic closure and cluster deletion”. In: *Algorithmica* 82.4 [2020], pp. 853–880. DOI: 10.1007/s00453-019-00617-1.

[82] Sylvain Guillemot, Frédéric Havet, Christophe Paul, and Anthony Perez. “On the (non-) existence of polynomial kernels for P_ℓ-free edge modification problems”. In: *Algorithmica* 65.4 [2013], pp. 900–926. DOI: 10.1007/s00453-012-9619-5.

[83] Jiong Guo. “A more effective linear kernelization for cluster editing”. In: *Theoretical Computer Science* 410.8-10 [2009], pp. 718–726. DOI: 10.1016/j.tcs.2008.10.021.

[84] Jiong Guo. "Problem kernels for NP-complete edge deletion problems: Split and related graphs". In: *International Symposium on Algorithms and Computation (ISAAC)*. Springer. 2007, pp. 915–926. DOI: 10.1007/978-3-540-77120-3_79.

[85] Jiong Guo, Christian Komusiewicz, Rolf Niedermeier, and Johannes Uhlmann. "A more relaxed model for graph-based data clustering: s-plex cluster editing". In: *SIAM Journal on Discrete Mathematics* 24.4 [2010], pp. 1662–1683. DOI: 10.1137/090767285.

[86] Michel Habib, Ross McConnell, Christophe Paul, and Laurent Viennot. "Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing". In: *Theoretical Computer Science* 234.1-2 [2000], pp. 59–84. DOI: 10.1016/S0304-3975(97)00241-7.

[87] Lars Hagen and Andrew B. Kahng. "New spectral methods for ratio cut partitioning and clustering". In: *IEEE transactions on computer-aided design of integrated circuits and systems* 11.9 [1992], pp. 1074–1085. DOI: 10.1109/43.159993.

[88] Rudolf Halin. "S-functions for graphs". In: *Journal of geometry* 8.1 [1976], pp. 171–186. DOI: 10.1007/BF01917434.

[89] Peter L. Hammer and Bruno Simeone. "The splittance of a graph". In: *Combinatorica* 1.3 [1981], pp. 275–284. DOI: 10.1007/BF02579333.

[90] Pierre Hansen and Brigitte Jaumard. "Cluster analysis and mathematical programming". In: *Mathematical programming* 79.1 [1997], pp. 191–215. DOI: 10.1007/BF02614317.

[91] Sepp Hartung, André Nichterlein, Rolf Niedermeier, and Ondřej Suchỳ. "A refined complexity analysis of degree anonymization in graphs". In: *Information and Computation* 243 [2015], pp. 249–262. DOI: 10.1016/j.ic.2014.12.017.

[92] Pavol Hell and Jing Huang. "Interval bigraphs and circular arc graphs". In: *Journal of Graph Theory* 46.4 [2004], pp. 313–327. DOI: 10.1002/jgt.20006.

[93] Wen-Lian Hsu and Tze-Heng Ma. "Fast and simple algorithms for recognizing chordal comparability graphs and interval graphs". In: *SIAM Journal on Computing* 28.3 [1998], pp. 1004–1020. DOI: 10.1137/S0097539792224814.

[94] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. "Which problems have strongly exponential complexity?" In: *Journal of Computer and System Sciences* 63.4 [2001], pp. 512–530. DOI: 10.1006/jcss.2001.1774.

[95] Haim Kaplan and Ron Shamir. "Bounded degree interval sandwich problems". In: *Algorithmica* 24.2 [1999], pp. 96–104. DOI: 10.1007/PL00009277.

[96] Haim Kaplan, Ron Shamir, and Robert E. Tarjan. "Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs". In: *SIAM Journal on Computing* 28.5 [1999], pp. 1906–1922. DOI: 10.1137/S0097539796303044.

[97] Ken-ichi Kawarabayashi and Bruce Reed. "An (almost) linear time algorithm for odd cycles transversal". In: *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms (SODA)*. SIAM. 2010, pp. 365–378.

[98] S. Thomas Kelly and Michael A. Black. “graphsim: An R package for simulating gene expression data from graph structures of biological pathways”. In: *Journal of Open Source Software* 5.51 [2020], p. 2161. DOI: 10.1101/2020.03.02.972471.

[99] Christian Komusiewicz and Johannes Uhlmann. “Alternative parameterizations for cluster editing”. In: *International Conference on Current Trends in Theory and Practice of Computer Science*. Springer. 2011, pp. 344–355. DOI: 10.1007/978-3-642-18381-2_29.

[100] Christian Komusiewicz and Johannes Uhlmann. “Cluster editing with locally bounded modifications”. In: *Discrete Applied Mathematics* 160.15 [2012], pp. 2259–2270. DOI: 10.1016/j.dam.2012.05.019.

[101] Athanasios L. Konstantinidis, Stavros D. Nikolopoulos, and Charis Papadopoulos. “Strong triadic closure in cographs and graphs of low maximum degree”. In: *Theoretical Computer Science* 740 [2018], pp. 76–84. DOI: 10.1016/j.tcs.2018.05.012.

[102] Norbert Korte and Rolf H. Möhring. “An incremental linear-time algorithm for recognizing interval graphs”. In: *SIAM Journal on Computing* 18.1 [1989], pp. 68–81. DOI: 10.1137/0218005.

[103] Stefan Kratsch. “Recent developments in kernelization: A survey”. In: *Bulletin of EATCS* 2.113 [2014].

[104] Stefan Kratsch, Marcin Pilipczuk, Michał Pilipczuk, and Magnus Wahlström. “Fixed-parameter tractability of multicut in directed acyclic graphs”. In: *SIAM Journal on Discrete Mathematics* 29.1 [2015], pp. 122–144. DOI: 10.1137/120904202.

[105] Stefan Kratsch and Magnus Wahlström. “Compression via matroids: a randomized polynomial kernel for odd cycle transversal”. In: *ACM Transactions on Algorithms* 10.4 [2014], pp. 1–15. DOI: 10.1145/2635810.

[106] Stefan Kratsch and Magnus Wahlström. “Representative sets and irrelevant vertices: New tools for kernelization”. In: *Journal of the ACM* 67.3 [2020], pp. 1–50. DOI: 10.1145/3390887.

[107] Stefan Kratsch and Magnus Wahlström. “Two edge modification problems without polynomial kernels”. In: *Discrete Optimization* 10.3 [2013], pp. 193–199. DOI: 10.1016/j.disopt.2013.02.001.

[108] Mirko Křivánek and Jaroslav Morávek. “NP-hard problems in hierarchical-tree clustering”. In: *Acta informatica* 23.3 [1986], pp. 311–323. DOI: 10.1007/BF00289116.

[109] C. Lekkeikerker and Johan Boland. “Representation of a finite graph by a set of intervals on the real line”. In: *Fundamenta Mathematicae* 51 [1962], pp. 45–64.

[110] John M. Lewis and Mihalis Yannakakis. “The node-deletion problem for hereditary properties is NP-complete”. In: *Journal of Computer and System Sciences* 20.2 [1980], pp. 219–230. DOI: 10.1016/0022-0000(80)90060-4.

[111] Min Chih Lin and Jayme L. Szwarcfiter. “Characterizations and linear time recognition of Helly circular-arc graphs”. In: *International Computing and Combinatorics Conference (COCOON)*. Springer. 2006, pp. 73–82. DOI: 10.1007/11809678_10.

[112] Yunlong Liu, Jianxin Wang, Jie You, Jianer Chen, and Yixin Cao. “Edge deletion problems: Branching facilitated by modular decomposition”. In: *Theoretical Computer Science* 573 [2015], pp. 63–70. DOI: 10.1016/j.tcs.2015.01.049.

[113] Daniel Lokshtanov. “New methods in parameterized algorithms and complexity”. In: *University of Bergen, Norway* [2009].

[114] Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. “Kernelization-Preprocessing with a Guarantee”. In: *The Multivariate Algorithmic Revolution and Beyond* 7370 [2012], pp. 129–161. DOI: 10.1007/978-3-642-30891-8_10.

[115] Frédéric Maffray and Myriam Preissmann. “Linear recognition of pseudo-split graphs”. In: *Discrete Applied Mathematics* 52.3 [1994], pp. 307–312. DOI: 10.1016/0166-218X(94)00022-0.

[116] Federico Mancini. “Graph modification problems related to graph classes”. In: *PhD degree dissertation, University of Bergen Norway* 2 [2008].

[117] Dániel Marx and R.B. Sandeep. “Incompressibility of H-free edge modification problems: Towards a dichotomy”. In: *Journal of Computer and System Sciences* 125 [2022], pp. 25–58. DOI: 10.1016/j.jcss.2021.11.001.

[118] Alireza R. Mashaghi, Abolfazl Ramezanpour, and Vahid Karimipour. “Investigation of a protein complex network”. In: *The European Physical Journal B-Condensed Matter and Complex Systems* 41.1 [2004], pp. 113–121. DOI: 10.1140/epjb/e2004-00301-0.

[119] Luke Mathieson and Stefan Szeider. "Editing graphs to satisfy degree constraints: A parameterized approach". In: *Journal of Computer and System Sciences* 78.1 [2012], pp. 179–191. DOI: 10.1016/j.jcss.2011.02.001.

[120] Hiroshi Nagamochi. "An approximation for finding a smallest 2-edge-connected subgraph containing a specified spanning tree". In: *Discrete Applied Mathematics* 126.1 [2003], pp. 83–113. DOI: 10.1016/S0166-218X(02)00218-4.

[121] James Nastos and Yong Gao. "Familial groups in social networks". In: *Social Networks* 35.3 [2013], pp. 439–450. DOI: 10.1016/j.socnet.2013.05.001.

[122] Assaf Natanzon. *Complexity and approximation of some graph modification problems*. University of Tel-Aviv, 1999.

[123] Assaf Natanzon, Ron Shamir, and Roded Sharan. "A polynomial approximation algorithm for the minimum fill-in problem". In: *SIAM Journal on Computing* 30.4 [2000], pp. 1067–1079. DOI: 10.1137/S0097539798336073.

[124] Assaf Natanzon, Ron Shamir, and Roded Sharan. "Complexity classification of some edge modification problems". In: *Discrete Applied Mathematics* 113.1 [2001], pp. 109–128. DOI: 10.1016/S0166-218X(00)00391-7.

[125] George L. Nemhauser and Leslie E. Trotter Jr. "Vertex packings: structural properties and algorithms". In: *Mathematical Programming* 8.1 [1975], pp. 232–248. DOI: 10.1007/BF01580444.

[126] Rolf Niedermeier. *Invitation to fixed-parameter algorithms*. Vol. 31. OUP Oxford, 2006.

[127] Rolf Niedermeier and Peter Rossmanith. “Upper bounds for Vertex Cover further improved”. In: *Lecture notes in computer science* [1999], pp. 561–570. DOI: 10.1007/3-540-49116-3_53.

[128] Naomi Nishimura, Prabhakar Ragde, and Dimitrios M. Thilikos. “On graph powers for leaf-labeled trees”. In: *Journal of Algorithms* 42.1 [2002], pp. 69–108. DOI: 10.1006/jagm.2001.1195.

[129] Stephan Olariu. “Paw-free graphs”. In: *Information Processing Letters* 28.1 [1988], pp. 53–54. DOI: 10.1016/0020-0190(88)90143-3.

[130] Seymour Parter. “The use of linear graphs in Gauss elimination”. In: *SIAM review* 3.2 [1961], pp. 119–130. DOI: 10.1137/1003021.

[131] Sheng-Lung Peng and Chi-Kang Chen. “On the interval completion of chordal graphs”. In: *Discrete Applied Mathematics* 154.6 [2006], pp. 1003–1010. DOI: 10.1016/j.dam.2005.09.010.

[132] Fábio Protti, Maise Dantas da Silva, and Jayme Luiz Szwarcfiter. “Applying modular decomposition to parameterized cluster editing problems”. In: *Theory of Computing Systems* 44.1 [2009], pp. 91–104. DOI: 10.1007/s00224-007-9032-7.

[133] Fábio Protti, Maise Dantas da Silva, and Jayme Luiz Szwarcfiter. “Applying modular decomposition to parameterized bicluster editing”. In: *International Workshop on Parameterized and Exact Computation (IPEC)*. Springer. 2006, pp. 1–12. DOI: 10.1007/11847250_1.

[134] Satish Rao and Andréa W. Richa. “New approximation techniques for some linear ordering problems”. In: *SIAM Journal on Computing* 34.2 [2005], pp. 388–404. DOI: 10.1137/S0097539702413197.

[135] Bruce Reed, Kaleigh Smith, and Adrian Vetta. “Finding odd cycle transversals”. In: *Operations Research Letters* 32.4 [2004], pp. 299–301. DOI: 10.1016/j.orl.2003.10.009.

[136] Romeo Rizzi, Vineet Bafna, Sorin Istrail, and Giuseppe Lancia. “Practical algorithms and fixed-parameter tractability for the single individual SNP haplotyping problem”. In: *International Workshop on Algorithms in Bioinformatics (WABI)*. Springer. 2002, pp. 29–43. DOI: 10.1007/3-540-45784-4_3.

[137] Fred S. Roberts. “Indifference graphs”. In: *Proof techniques in graph theory* [1969], pp. 139–146.

[138] Neil Robertson and Paul D. Seymour. “Graph minors. III. Planar tree-width”. In: *Journal of Combinatorial Theory, Series B* 36.1 [1984], pp. 49–64. DOI: 10.1016/0095-8956(84)90013-3.

[139] Donald J. Rose. “A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations”. In: *Graph theory and computing*. Elsevier, 1972, pp. 183–217. DOI: 10.1016/B978-1-4832-3187-7.50018-0.

[140] Donald J. Rose and Robert E. Tarjan. “Algorithmic aspects of vertex elimination on directed graphs”. In: *SIAM Journal on Applied Mathematics* 34.1 [1978], pp. 176–197. DOI: 10.1137/0134014.

[141] R.B. Sandeep and Naveen Sivadasan. “Parameterized lower bound and improved kernel for diamond-free edge deletion”. In: *10th International Symposium on Parameterized and Exact Computation (IPEC)*. Schloss

Dagstuhl-Leibniz-Zentrum fuer Informatik. 2015. DOI: 10.4230/LIPIcs.IPEC.2015.365.

[142] Alexander Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

[143] Preya Shah, Arian Ashourvan, Fadi Mikhail, Adam Pines, Lohith Kini, Kelly Oechsel, Sandhitsu R. Das, Joel M. Stein, Russell T. Shinohara, Danielle S. Bassett, et al. “Characterizing the role of the structural connectome in seizure dynamics”. In: *Brain* 142.7 [2019], pp. 1955–1972. DOI: 10.1093/brain/awz125.

[144] Ron Shamir, Roded Sharan, and Dekel Tsur. “Cluster graph modification problems”. In: *Discrete Applied Mathematics* 144.1-2 [2004], pp. 173–182. DOI: 10.1016/j.dam.2004.01.007.

[145] Roded Sharan. “Graph modification problems and their applications to genomic research”. PhD thesis. Tel-Aviv University, 2002.

[146] Roded Sharan, Adi Maron-Katz, and Ron Shamir. “CLICK and EXPANDER: a system for clustering and visualizing gene expression data”. In: *Bioinformatics* 19.14 [2003], pp. 1787–1799. DOI: 10.1093/bioinformatics/btg232.

[147] Gerard Sierksma and Yori Zwols. *Linear and integer optimization: theory and practice*. CRC Press, 2015.

[148] Ulrike Stege and Michael Ralph Fellows. “An improved fixed parameter tractable algorithm for vertex cover”. In: *Technical report/Departement Informatik, ETH Zürich* 318 [1999]. DOI: 10.3929/ethz-a-006653305.

[149] Földes Stéphane and Peter L. Hammer. “Split graphs”. In: *Proceedings of the Eighth Southeastern Conference on Combinatorics, Graph Theory and Computing (SEICCGTC)*. Vol. XIX. 1977, pp. 311–315.

[150] Kazuhiko Takamizawa, Takao Nishizeki, and Nobuji Saito. “Linear-time computability of combinatorial problems on series-parallel graphs”. In: *Journal of the ACM* 29.3 [1982], pp. 623–641. DOI: 10.1145/322326.322328.

[151] William T. Trotter Jr and John I. Moore Jr. “Characterization problems for graphs, partially ordered sets, lattices, and families of sets”. In: *Discrete Mathematics* 16.4 [1976], pp. 361–381. DOI: 10.1016/S0012-365X(76)80011-8.

[152] Dekel Tsur. “Kernel for K_t-free edge deletion”. In: *Information Processing Letters* 167 [2021], p. 106082. DOI: 10.1016/j.ipl.2020.106082.

[153] Alan Tucker. “Characterizing circular-arc graphs”. In: *Bull. Amer. Math. Soc.* 76.4 [1970], pp. 1257–1260. DOI: 10.1090/S0002-9904-1970-12628-3.

[154] Alan Tucker. “Structure theorems for some circular-arc graphs”. In: *Discrete Mathematics* 7.1-2 [1974], pp. 167–195. DOI: 10.1016/S0012-365X(74)80027-0.

[155] RI Tyshkevich, OI Melnikow, and VM Kotov. “On graphs and degree sequences: the canonical decomposition”. In: *Cybern Syst Anal* 17 [1981], pp. 722–727. DOI: 10.1007/BF01069217.

[156] Yngve Villanger, Pinar Heggernes, Christophe Paul, and Jan Arne Telle. "Interval completion is fixed parameter tractable". In: *SIAM Journal on Computing* 38.5 [2009], pp. 2007–2020. DOI: 10.1137/070710913.

[157] Toshimasa Watanabe and Akira Nakamura. "Edge-connectivity augmentation problems". In: *Journal of Computer and System Sciences* 35.1 [1987], pp. 96–144. DOI: 10.1016/0022-0000(87)90038-9.

[158] Gerd Wegner. *Eigenschaften der Nerven homologisch-einfacher Familien im R^n*. éditeur non identifié, 1967.

[159] David P. Williamson and David B. Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

[160] Gerhard J. Woeginger. "Exact algorithms for NP-hard problems: A survey". In: *Combinatorial optimization—eureka, you shrink!* Springer, 2003, pp. 185–207. DOI: 10.1007/3-540-36478-1_17.

[161] Elliot S. Wolk. "A note on "The comparability graph of a tree"". In: *Proceedings of the American Mathematical Society* 16.1 [1965], pp. 17–20. DOI: 10.1090/S0002-9939-1965-0172274-5.

[162] Elliot S. Wolk. "The comparability graph of a tree". In: *Proceedings of the American Mathematical Society* 13.5 [1962], pp. 789–795. DOI: 10.2307/2034179.

[163] Zhenyu Wu and Richard Leahy. "An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15.11 [1993], pp. 1101–1113. DOI: 10.1109/34.244673.

[164] Ge Xia and Yong Zhang. "Kernelization for cycle transversal problems". In: *Discrete Applied Mathematics* 160.7-8 [2012], pp. 1224–1231. DOI: 10.1016/j.dam.2011.12.024.

[165] Jing-Ho Yan, Jer-Jeong Chen, Gerard J. Chang, et al. "Quasi-threshold graphs". In: *Discrete Applied Mathematics* 69.3 [1996], pp. 247–255. DOI: 10.1016/0166-218X(96)00094-7.

[166] Mihalis Yannakakis. "Computing the minimum fill-in is NP-complete". In: *SIAM Journal on Algebraic Discrete Methods* 2.1 [1981], pp. 77–79. DOI: 10.1137/0602010.

[167] Mihalis Yannakakis. "Node-and edge-deletion NP-complete problems". In: *Proceedings of the tenth annual ACM symposium on Theory of computing (STOC)*. 1978, pp. 253–264. DOI: 10.1145/800133.804355.

[168] Mihalis Yannakakis. "Node-deletion problems on bipartite graphs". In: *SIAM Journal on Computing* 10.2 [1981], pp. 310–327. DOI: 10.1137/0210022.

[169] Xiaotong Zhuang and Santosh Pande. "Resolving register bank conflicts for a network processor". In: *12th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE. 2003, pp. 269–278. DOI: 10.1109/PACT.2003.1238022.