# Graph editing by partial complements

Scholte, T.R.

*Award date:*
2018

# Graph Editing by Partial Complements

Timo R. Scholte

*Bergen, July 20, 2018*
Version: Final Draft

Eindhoven University of Technology

Department of Mathematics and Computer Science
Combinatorial Optimization Group (CO)

Master Thesis

# Graph Editing by Partial Complements

Timo R. Scholte

*Supervisors:* Jesper Nederlof
Eindhoven University of Technology

Fedor V. Fomin
University of Bergen

*Evaluation Comitee:* Hans L. Bodlaender
Eindhoven University of Technology

Bart M. P. Jansen
Eindhoven University of Technology

July 20, 2018

**Timo R. Scholte**
*Graph Editing by Partial Complements*
Master Thesis, July 20, 2018
Supervisors: Jesper Nederlof and Fedor V. Fomin
Evaluation Comitee: Hans L. Bodlaender and Bart M. P. Jansen

**Eindhoven University of Technology**
Department of Mathematics and Computer Science
De Zaale
5612 AJ, Eindhoven

# Abstract

The *partial complement*, introduced by Kamiński, Lozin, and Milanič [18], is a graph operator, that given a subset of the vertices, complements the subgraph induced on those vertices, whilst keeping the rest of the graph intact. Fomin et al. [14] considered this operator in a graph editing setting. They studied the question "For a given graph $G$ and graph family $\mathcal{F}$, is it possible to partially complement $G$ to $\mathcal{F}$?". They showed that this was polynomial time solvable for various graph families, and proved that this problem is NP-complete for $r$-regular graphs. Their scope only includes a single application of the partial complement operator.

We extend this problem by allowing multiple subsequent partial complements. Our goal is to find polynomial time algorithms, or fixed parameter tractable algorithms parameterized by the number of partial complements we allow, for various graph families. We show that a partial complement alters the rank-width of a graph by at most 1. This tells us that if we can express our problem in $\text{MSO}_1$ logic and our goal family has bounded rank-width, our problems are FPT parameterized by the number of partial complements we allow. We show that if $\mathcal{F}$ is the family of $H$-free simple graphs, this approach only works if $H$ is an induced subgraph of $P_4$.

We also pursue algorithms that do not rely on $\text{MSO}_1$ logic or rank-width. To achieve this, we distinguish between simple graphs and unigraphs (an extension of simple graphs where every vertex is allowed to have at most one loop). Since unigraphs are allowed to have loops, a partial complement on a unigraph will also add or remove loops. This has nice algebraic implications, which we can exploit to show that we can determine how many partial complements are required to turn a unigraph into an edgeless graph in time $O(n^\omega)$, where $\omega$ is the matrix multiplication exponent. We use this result to find FPT algorithms for unigraphs to $P_3^*$-free reflexive unigraphs of at most $d$ connected components, simple graphs to edgeless simple graphs, and simple graphs to $P_3$-free simple graphs of at most $d$ connected components, with running times $O(\frac{d^{2^k+d}}{d!}n^\omega)$, $O(2^{2^k}n^\omega + n^3)$, and $O(\frac{(2d)^{2^k+d}}{d!}n^\omega + n^3)$, respectively.

# List of Notation

We will give a list of notation of concepts that are used in multiple sections, along with short descriptions, and a reference to the section of the thesis in which this notation is defined.

- $\oplus$: The partial complement operator on simple graphs.

  $$G \oplus S := (V(G), E(G)\Delta\{\{u, v\} : \forall u, v \in S \text{ with } u \neq v\}).$$

  Defined in the introduction of Chapter 1.

- $\widehat{\oplus}$: The partial complement operator on unigraphs.

  $$G \widehat{\oplus} S := (V(G), E(G)\Delta\{\{u, v\} : \forall u, v \in S\}).$$

  Defined in the introduction of Chapter 1.

- $\mathrm{rk}\,(A)$: Matrix rank of $A$; The dimension of the row space (or column space) of $A$ over some field. In this thesis, we use the matrix rank over $GF[2]$, unless another field is specified. Defined in Section 2.2.

- $\mathrm{rk}_{\mathbb{S}}\,(A)$: The symmetric rank of $A$ over $GF[2]$; the minimal number $k$ such that there exists column vectors $u_1, \ldots, u_k$ such that $A = \sum_{i=1}^{k} u_i u_i^T$. Defined in the introduction of Chapter 4.

- $PC_{\mathcal{F}}(G)$: The problem of finding the minimal number $k$ of vertex subsets $S_1, \ldots, S_k \subseteq V(G)$ such that $(G \oplus S_1 \oplus \cdots \oplus S_k) \in \mathcal{F}$. Defined in Section 1.1.

- $|PC_{\mathcal{F}}(G)|$: The amount of vertex subsets in the minimal solution to $PC_{\mathcal{F}}(G)$. Defined in Section 1.1.

- $PC_{\mathcal{F}}(G, k)$: The decision variant of $PC_{\mathcal{F}}(G)$; Is $|PC_{\mathcal{F}}(G)| \leq k$? Defined in Section 1.1.

- $\widehat{PC_{\mathcal{F}}}(G)$: The problem of finding the minimal number $k$ of vertex subset $S_1, \ldots, S_k \subseteq V(G)$ such that $(G \widehat{\oplus} S_1 \widehat{\oplus} \cdots \widehat{\oplus} S_k) \in \mathcal{F}$. Defined in Section 1.1.

- $|\widehat{PC_{\mathcal{F}}}(G)|$: The amount of vertex subsets in the minimal solution to $\widehat{PC_{\mathcal{F}}}(G)$. Defined in Section 1.1.

- $\widehat{PC_{\mathcal{F}}}(G, k)$: The decision variant of $\widehat{PC_{\mathcal{F}}}(G)$; Is $|\widehat{PC_{\mathcal{F}}}(G)| \leq k$? Defined in Section 1.1.

- $P_3^*$: A path on 3 vertices where every vertex has a loop. Originally defined in Section 4.2.

- $\mathcal{G}$: The family of edgeless graphs. Defined in Section 4.1.

- $\mathcal{H}$: The family of $P_3$-free simple graphs. Defined in Section 4.2.

- $\mathcal{H}^d$: The family of $P_3$-free simple graphs consisting of at most $d$ connected components. Defined in Section 4.2.

- $\widehat{\mathcal{H}}$: The loop extension family of $\mathcal{H}$ consisting of $P_3^*$-free reflexive unigraphs. Defined in Section 4.2.

- $\widehat{\mathcal{H}}^d$: The loop extension family of $\mathcal{H}^d$ consisting of $P_3^*$-free reflexive unigraphs of at most $d$ connected components. Defined in Section 4.2.

- $\eta_G$: Given a simple graph $G$, $\eta_G$ is the minimum rank of $A_{\widehat{G}}$, over all possible $\widehat{G}$'s created by taking $G$ and adding loops. Originally defined in Section 5.1.

- $\mathcal{K}$: The family of critical cliques $K$ within a graph $G$ of cardinality $|K| \geq 2$. Originally defined in Section 5.1.

- $\mathcal{A}$: The family of critical independent sets $A$ within a graph $G$ of cardinality $|A| \geq 2$. Originally defined in Section 5.1.

- $\mathcal{S}$: The family of single vertex sets $S$ in some graph $G$, such that $\{u\} \in \mathcal{S}$ if and only if there is no $K \in \mathcal{K}$ and no $A \in \mathcal{A}$ for which $u \in K$ or $u \in A$. Originally defined in Section 5.1.

- $I_S$ for $S \subseteq \{1, \ldots, n\}$: $n \times n$ $GF[2]$ matrix where $I_S[i, j] = 1$ if and only if $i = j$ and $i \in S$. Originally defined in Section 5.1.

- $N_G(v) = \{u \mid u \in V(G), \{u, v\} \in E(G)\}$. Defined in Section 2.2.

- $N_G[v] = N_G(v) \cup \{v\}$. Defined in Section 2.2.
- $N_G[X] = \bigcup_{v \in X} N_G[v]$. Defined in Section 2.2.
- $N_G(X) = N_G[X] \setminus X$. Defined in Section 2.2.

# Contents

**Abstract** v

**List of Notation** vii

**1 Introduction** 1
- 1.1 Problem Description . . . 4
- 1.2 Overview of this Thesis . . . 4

**2 Preliminaries** 9
- 2.1 Short Introduction to Computational Complexity . . . 9
- 2.2 Basic Definitions . . . 12
- 2.3 Rank-Width . . . 14
- 2.4 Clique-Width . . . 15
- 2.5 Monadic Second Order Logic . . . 16

**3 FPT by Bounded Rank-Width** 19
- 3.1 Proving FPT . . . 20
- 3.2 Examples of Graph Families Satisfying the Constraints . . . 23

**4 Unigraphs and Symmetric Rank** 27
- 4.1 Edgeless Graphs . . . 28
  - 4.1.1 Approximation Algorithm . . . 29
  - 4.1.2 Zero-Diagonal Matrices . . . 36
  - 4.1.3 Further Results on Symmetric Rank . . . 46
- 4.2 $P_3^*$-Free Reflexive Unigraphs of at most $d$ Connected Components . . . 47

**5 Implications for Simple Graphs** 51
- 5.1 Edgeless Graphs . . . 52
- 5.2 $P_3$-Free Graphs of at most $d$ Connected Components . . . 58

**6 Conclusion and Discussion** 63

**7 Open Questions** 67

**Bibliography** 73

# 1 Introduction

A popular research topic within graph theory is that of “graph modification problems". In these problems, we are given a graph, some graph modification operator, and a goal graph family. The question is then “What is the minimum amount of operations required to turn our graph into a graph that is part of our goal family". The most common graph operators considered for graph modification problems are a combination of vertex deletion, edge deletion, and edge insertion. However, other operators can also be considered.

For instance, various results have been obtained for ‘complementation operators’. Recall that $\bar{G}$, the complement of a graph $G$, is a graph on the same vertices such that two distinct vertices are adjacent in $\bar{G}$ if and only if they are not adjacent in $G$. An interesting operator is the *Seidel switch* [26]. This operator complements the adjacencies of a vertex $v$, i.e. it removes the edges between $v$ and its neighbours, while adding edges between $v$ and its non-neighbours. The Seidel switch has been studied in a graph modification setting [12, 13], with an emphasis on existence of a sequence of operations, rather than minimizing the amount of operations. Another interesting operator of this type is the *local complementation*. For a vertex $v$ of a graph $G$, the local complementation of $G$ at $v$ is the graph obtained by complementing the neighbourhood of $v$. In other words, it adds edges between any two distinct non-adjacent neighbours of $v$ and removes edges between any two distinct adjacent neighbours of $v$. This operation plays a crucial role in the definition of *vertex-minors* [22]. Various algorithmic results concerning local complementation have been found [2, 18].

An interesting graph modification problem is the *cluster editing* problem. Within cluster editing, we are given a graph $G$, and are asked how many edge insertions and deletions are required to turn $G$ into a cluster graph, i.e. a graph in which every connected component is a clique. In practice, this problem is used to remove inconsistencies from a data set. If vertices represent data points, and edges represent some particular correlation between these data points, we might expect our data set to be a cluster graph. Therefore, we can expect that the edges we insert and delete are the inconsistencies in our data. The cluster editing problem has applications in many fields, including machine learning, data-mining, information retrieval, and computational biology [4].

Let us briefly explain the concept of *fixed parameter tractability* (FPT), with a more complete explanation in Section 2.1, which also contains an explanation for the other complexity classes used throughout this thesis. Recall that if a problem is NP-complete, you are unlikely to find a polynomial time algorithm for this problem as this would prove that P=NP. This is where FPT steps in. If our problem instance comes with some kind of parameter $k$, which denotes some information about the problem instance, we could use this to find an efficient algorithm. A problem is said to be in FPT if it allows an algorithm with running time $O(f(k)n^c)$ for some function $f$ and some constant $c$. The beauty of this is that if we regard $k$ as a constant, this gives us a polynomial time algorithm.

A common technique in developing FPT algorithms is *kernelization*. We take our problem instance $(I, k)$, and our parameter $k$, and start preprocessing our problem instance $(I, k)$ until we find a new problem instance $(I', k')$ with $k' \leq k$, such that $(I', k')$ is a yes-instance if and only if $(I, k)$ is, and the size of $I'$ is bounded by a function of $k'$. It is important that the preprocessing is done in polynomial time. For a more thorough explanation of this concept, read Section 2.1.

Cluster editing is a typical example for FPT. In the applications we have discussed, we can usually assume that the amount of 'inconsistencies' is not too big. Furthermore, it is relatively easy to show that cluster editing permits a kernel of size $O(k^2)$[10]. Currently, the best known kernel for cluster editing gives us an instance of at most $2k$ vertices [4]. This kernel is achieved using the notion of critical cliques, which will also play a role later in this thesis. Currently, the fastest known algorithm that solves cluster editing for a graph with $n$ vertices and $m$ edges runs in $O(1.62^k + m + n)$ [1].

Within this project, we set out to study a generalization of the cluster editing problem using a 'complementation operator'. Namely, cluster editing by 'partial complements', where we want to know how many partial complements are required to turn our graph into a cluster graph. The partial complement is a graph operator, introduced by Kamiński, Lozin, and Milanič [18] under the name 'subgraph complementation', that is defined as follows:

**Definition.** Given a simple graph $G = (V, E)$, let $S \subseteq V$ be a subset of vertices. The *partial complement on* $S$, denoted by $G \oplus S$, returns a simple graph $G' = (V, E')$ such that:

$$E' = E \Delta \{\{u, v\} : \forall u, v \in S \text{ with } u \neq v\}$$

Alternatively, we could define $E'$ in the following manner.

$$\{u, v\} \in E' \Leftrightarrow \{u, v\} \notin E \qquad \forall u, v \in S \text{ with } u \neq v$$
$$\{u, v\} \in E' \Leftrightarrow \{u, v\} \in E \qquad \forall u, v \text{ not both in } S$$

The idea is that we complement all edges restricted to a small part of our graph. If we restrict the size of the partial complements to two vertices, the problem is identical to traditional cluster editing. If we allow partial complements of arbitrary sizes, this is no longer the case. Over the course of the project, we decided to generalize our problem even further; we are not only interested in graph editing by partial complements towards cluster graphs, but we consider various graph families as our goal.

The partial complement has been studied in a graph modification setting before [14]. It is important to note that the existing work limits itself to a single application of the partial complement operator, whereas we shall focus on multiple successive applications. However, some of the known results still have interesting implications for our problem. For instance, Fomin et al. [14] showed that deciding whether a graph $G$ can be turned into an $r$-regular graph with one partial complement is $NP$-complete. This implies that our problem of deciding whether or not a graph can be turned into a graph that is part of some graph family with at most $k$ partial complements will be $NP$-complete for some graph families. Furthermore, Fomin et al. [14] showed that if $\mathcal{F}$ is a family of graphs of bounded clique-width that is expressible in $\text{MSO}_1$, then deciding whether a graph $G$ has a vertex subset $S \subset V(G)$ such that $G \oplus S \in \mathcal{F}$ can be solved in polynomial time. In Chapter 3, we extend these results towards our setting.

Over the course of this project, we discovered it was relevant to also look into partial complements on unigraphs, i.e. partial complements on graphs where loops are permitted. As loops are permitted, the definition of a partial complement changes:

**Definition.** Given a unigraph $G = (V, E)$, let $S \subseteq V$ be a subset of vertices. The *partial complement on* $S$, denoted by $G\widehat{\oplus}S$, returns a unigraph $G' = (V, E')$ such that:

$$E' = E\Delta\{\{u, v\} : \forall u, v \in S\}$$

Alternatively, we could define $E'$ in the following manner.

$$\begin{aligned} \{u, v\} \in E' \Leftrightarrow \{u, v\} \notin E \qquad & \forall u, v \in S \\ \{u, v\} \in E' \Leftrightarrow \{u, v\} \in E \qquad & \forall u, v \text{ not both in } S \end{aligned}$$

Note that we use different notation for the different definitions of a partial complement to emphasize whether the graph we are working on is a unigraph or a simple graph. The difference between these two definitions is that on unigraphs, whilst taking a partial complement on $S$, we add/remove loops at all vertices of $S$. As we will see in Chapter 4, working with the partial complement on unigraphs has many advantages compared to working with the partial complement on simple graphs.

## 1.1 Problem Description

We want to study graph modification problems where we are allowed to use multiple partial complements to turn a graph into a graph belonging to a particular family of graphs. More formally:

**Definition.** Let $\mathcal{F}$ be a graph family, and $G$ be a simple graph. Define $PC_{\mathcal{F}}(G)$ as the problem of finding the minimal number $l$ of vertex subsets $S_1, \ldots, S_l \subseteq V(G)$ such that $(G \oplus S_1 \oplus \cdots \oplus S_l) \in \mathcal{F}$. We write $|PC_{\mathcal{F}}(G)| = l$. Alternatively, we might look into the decision variant $PC_{\mathcal{F}}(G, k)$, where we decide whether or not $|PC_{\mathcal{F}}(G)| \leq k$.

Our goal for this project is to find FPT algorithms for $PC_{\mathcal{F}}(G, k)$ for certain graph families. Our focus will lie on families of $H$-free graphs, where $H$ is some small graph. In particular, we use $P_2, \overline{P_2}, P_3, \overline{P_3}$, and $P_4$ as candidates for $H$. Note that cluster graphs and $P_3$-free graphs are the same thing. We will also look into $P_3$-free graphs with a bound on the number of connected components.

Of course, we also consider the problem's analogue for unigraphs:

**Definition.** Let $\mathcal{F}$ be a graph family, and $G$ be a unigraph. Define $\widehat{PC_{\mathcal{F}}}(G)$ as the problem of finding the minimal number $l$ of vertex subsets $S_1, \ldots, S_l \subseteq V(G)$ such that $(G \widehat{\oplus} S_1 \widehat{\oplus} \cdots \widehat{\oplus} S_l) \in \mathcal{F}$. We write $|\widehat{PC_{\mathcal{F}}}(G)| = l$. Alternatively, we might look into the decision variant $\widehat{PC_{\mathcal{F}}}(G, k)$, where we decide whether or not $|\widehat{PC_{\mathcal{F}}}(G)| \leq k$.

For $\widehat{PC_{\mathcal{F}}}(G)$, our goals are similar to our goals for $PC_{\mathcal{F}}(G)$. We aim to prove the problem is FPT for certain graph families. Specifically, we consider edgeless graphs, and $P_3^*$-free reflexive unigraphs of at most $d$ components as the goal families.

## 1.2 Overview of this Thesis

Let us summarize the structure and the results of this thesis. In Chapter 2, we will present definitions of concepts used throughout the remainder of this thesis that we expect our reader to be familiar with.

In Chapter 3, we will prove the following theorem:

**Theorem 3.2.** *Let $\mathcal{F}$ be a family of graphs such that there exists a constant $c$ for which $G \in \mathcal{F}$ implies rw$(G) \leq c$, and the decision problem $PC_{\mathcal{F}}(G, k)$ can be expressed with an MSO$_1$ formula. Then $PC_{\mathcal{F}}(G, k)$ is FPT parametrized by $k$.*

This proves that our problem is FPT for various graph families. In Section 3.2, we show that a family of $H$-free simple graphs satisfies these constraints if and only if $H$ is an induced subgraph of $P_4$.

The FPT algorithm corresponding to Theorem 3.2 is implied by a theorem by Courcelle, Makowsky, and Rotics[8], stated in this thesis as Theorem 3.3. Unfortunately, the bound on the running time implied by this theorem is notoriously large. Because of this large running time, we try to find better algorithms for two graph families, namely $\mathcal{G}$, the family of edgeless graphs, and $\mathcal{H}^d$, the family of $P_3$-free simple graphs consisting of at most $d$ connected components.

To achieve these algorithms, we first consider an alternate version of our problem, which we do in Chapter 4. We alter our focus from simple graphs to unigraphs, i.e. graphs that may include loops. This allows us to utilize a more algebraic approach. Furthermore, we introduce the notion of symmetric rank of a matrix $A$, the minimal amount of symmetric rank one matrices that sum up to $A$. We show that $\widehat{PC_{\mathcal{G}}}(G)$ is equal to $\mathrm{rk}_{\mathbb{S}}(A_G)$, where $\mathcal{G}$ denotes the family of edgeless graphs, $\mathrm{rk}_{\mathbb{S}}(A_G)$ denotes the symmetric rank of $A_G$, and $A_G$ denotes the adjacency matrix of $G$. Moreover, we prove a strong relation between normal rank and symmetric rank:

**Theorem 4.1.** *Let $A$ be a symmetric $GF[2]$ matrix, such that $A$ is not the all zero matrix. If $A$ has no $1$ on its diagonal, then $\mathrm{rk}_{\mathbb{S}}(A) = \mathrm{rk}(A) + 1$. Otherwise, $\mathrm{rk}_{\mathbb{S}}(A) = \mathrm{rk}(A)$.*

This result implies an $O(n^\omega)$ time algorithm for editing (on unigraphs) towards $\mathcal{G}$, where $\omega$ denotes the matrix multiplication constant. The following are some of the intermediate results proved whilst proving Theorem 4.1.

**Lemma 4.7.** *Given symmetric $GF[2]$ matrix $A$, with $\mathrm{rk}(A) = r > 0$, such that all diagonal entries of $A$ are 0's. There exists no symmetric $GF[2]$ rank one matrix $B$ for which $\mathrm{rk}(A + B) = r - 1$.*

**Lemma 4.8.** *Given symmetric $GF[2]$ matrix $A$, such that all diagonal entries of $A$ are $0$. Then $\mathrm{rk}(A)$ is even.*

**Lemma 4.10.** *Let $A$ be a symmetric $GF[2]$ matrix with at least one 1 on its diagonal. Let $a_1, \ldots, a_n$ denote the $n$ columns of $A$. Let $I$ bet the indices of the rows of $A$ that have a 1 on their diagonal position. If $A + a_i a_i^T$ is a zero-diagonal matrix for all $i \in I$, then $\mathrm{rk}_{\mathbb{S}}(A) = \mathrm{rk}(A)$.*

Furthermore, within Section 4.2, we consider $\widehat{PC_{\widehat{\mathcal{H}}^d}}(G)$, where $\widehat{\mathcal{H}}^d$ is the family of $P_3^*$-free reflexive unigraphs of at most $d$ connected components. We consider how we can translate such problem instances towards instances of the form $\widehat{PC_{\mathcal{G}}}(G')$.

This leads us towards an $O\left(\frac{d^{2^{k+d}}}{d!}n^{\omega}\right)$ time algorithm for editing unigraphs towards $\widehat{\mathcal{H}}^d$.

In Chapter 5, we try to translate the results of Chapter 4 back to simple graphs. Specifically, we look into how we can turn a simple graph into a unigraph, so that the solutions to the partial complement problem is the same in both graphs. This is motivated by the following lemma:

**Lemma 5.2.** *Let $\mathcal{G}$ be the family of edgeless graphs. Given a simple graph $G = (V, E)$ with $E \neq \emptyset$, it holds that*

$$|PC_{\mathcal{G}}(G)| = \min_{L \subseteq V}\{\text{rk}\left(A_{\widehat{G}}\right) : \widehat{G} = (V, E \cup \{\{u, u\} \mid \forall u \in L\}), L \neq \emptyset\}$$

We define the minimization problem on the right hand side of the equation above as $RMDA(A_G)$. We then consider critical cliques and critical independent sets, which are vertex subsets $X \subseteq V(G)$, such that $\forall v \in X$ it holds that $N_G[v] = N_G[X]$ (for critical cliques) or $N_G(v) = N_G(X)$ (for critical independent sets). We are able to produce the following intermediate results:

**Lemma 5.4.** *Given a simple graph $G$. Let $\mathcal{K}$ be the family of all critical cliques $K$ of $G$ with $|K| \geq 2$, $\mathcal{A}$ be the family of all critical independent sets $A$ of $G$ with $|A| \geq 2$, and $\mathcal{S}$ be the family of one element subsets of $V(G)$, containing each vertex that is not covered by either $\mathcal{K}$ or $\mathcal{A}$ as a separate subset. We then know that every vertex $v$ appears in exactly one element of $\mathcal{K} \cup \mathcal{A} \cup \mathcal{S}$, and that $|\mathcal{K}| + |\mathcal{A}| + |\mathcal{S}| \leq 2^{RMDA(A_G)}$.*

**Lemma 5.5.** *Let $G$ be a simple graph, let $L^*$ be the vertex subset corresponding to an optimal solution to $\text{RMDA}(A_G)$, and let $X$ be a critical clique or critical independent set. It then holds that $X \subseteq L^*$ or $X \cap L^* = \emptyset$.*

These two lemmas lead us towards an $O\left(2^{2^k}n^{\omega} + n^3\right)$ time algorithm for $PC_{\mathcal{G}}(G, k)$.

After this, we focus on $\mathcal{H}^d$, the family of $P_3$-free simple graphs of at most $d$ components. We prove an analogue for Lemma 5.5:

**Lemma 5.9.** *Let $G$ be an instance of $PC_{\mathcal{H}^d}$, and let $\mathcal{K}$ and $\mathcal{A}$ be the critical cliques and critical independent sets of $G$ of size larger than 1. For a solution $S_1, \ldots, S_k$, we define $L \subseteq V(G)$ as*

$$L = \{v \mid v \in V(G) \text{ for which } |\{i \mid v \in S_i\}| \text{ is even}\}$$

*There exists an optimal solution $S_1, \ldots, S_k$ and corresponding $L$ such that $\forall X \in \mathcal{K} \cup \mathcal{A}$, it holds that $X \subseteq L$ or $X \cap L = \emptyset$.*

This brings us to an $O\left(\frac{(2d)^{2^{k+d}}}{d!}n^{\omega}+n^3\right)$ time algorithm for $PC_{\mathcal{H}^d}$.

Finally, Chapter 6 is reserved for conclusion and discussion of the project, and after that, we pose some open questions related to this thesis in Chapter 7.

# 2 Preliminaries

In this chapter we will state the preliminaries of this thesis. We will start by briefly explaining computational complexity, with a focus on $P$, $NP$, $NP$-complete, and FPT. After this, we give some basic definitions from graph theory, set theory and (linear) algebra. Finally, we will give a simplified explanation for rank-width, clique-width, and monadic second order logic. For the remainder of the thesis, we assume that the reader is familiar with all of the concepts described within this chapter.

## 2.1 Short Introduction to Computational Complexity

A core concept within this thesis is that of computational complexity. Within this subsection, we will describe the following complexity classes: P, NP, NP-complete, and $FPT$. If the reader is already familiar with these classes, this section can be skipped. For a more thorough explanation on these classes, we recommend the books *Introduction to Algorithms* [7] for a description of P, NP, NP-completeness, and $O$-notation; and *Parameterized Algorithms* [10] for a description of $FPT$ and any related concepts.

When designing an algorithm, it is relevant to know how long the execution of your algorithm will take. The exact running time will depend on the machine the algorithm runs on, as well as the exact instance of the problem. This means that there can be a lot of variation in running time between two runs of the same algorithm.

Though the running time for one instance on two different machines may vary, we know that in a deterministic algorithm both machines must take the same number of computational steps to arrive at a solution. Therefore, it is more interesting to look at the amount of steps required to find a solution, as this can be expressed independently of the machine the algorithm runs on.

Now we need a way to express the amount of steps needed, without knowing what instance the algorithm will run on. This is done by expressing the number of steps as function of the size of the input instance, which is often denoted by $n$. Of course,

the exact number of steps might depend on the structure of the instance, and can therefore not always be described as a function of the input size. Because of this, we are more interested in the asymptotic behaviour of our amount of steps when the input size $n$ becomes large, and specifically the amount of steps required to solve the 'worst-case' instance of size $n$. This worst-case refers to the fact that this instance needs the maximum amount of steps of all instances of this size. To express this, we use $O$-notation:

**Definition** (Cormen et al. [7])**.** For a given function $g(n)$ we denote $O(g(n))$ as the set of functions

$$O(g(n)) = \{f(n) : \text{there exists positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0\}.$$

Note that there exists different notations for asymptotic lower bounds, and combinations of asymptotic higher and lower bounds. However, we do not use these within this project, and shall therefore not describe them. Using $O$-notation, we can create insight in the relation between the instance size, and the running time of our algorithm.

Ideally, we want to find an algorithm running in time $O(g(n))$, for as small a $g(n)$ as possible. Usually, we want $g(n)$ to be a polynomial, as the growth of polynomial functions is significantly smaller than that of other functions that turn up naturally in describing the number of steps, such as an exponential function. If $g(n)$ is a polynomial, we say that we have a polynomial time algorithm to solve our problem. It can be hard to be determine whether or not there exists a polynomial time algorithm for some problems. To cope with this fact, the notion of NP-complete problems was introduced [6, 19]. To understand what this means, let us first introduce the complexity classes P and NP.

The class of problems P consists of the problems for which some polynomial time algorithm exists. The class of NP consists of the problems that are verifiable in polynomial time. This means we can create a certificate for the correctness of a 'yes' solution, such that we can check in polynomial time that this certificate is correct.

Note that any problem in P is also in NP, as we can verify any solution in polynomial time, even without a certificate, because there exists some polynomial time algorithm that solves the problem. It is an open question whether or not any problem in NP is also in P.

This brings us to the NP-complete class. Informally, a problem is NP-complete if it is in NP and as "hard" as any other problem in NP. What this implies is that if

some NP-complete problem is shown to have a polynomial time algorithm, then all problems within NP permit a polynomial time algorithm. To show that some problem $A$ in NP is NP-complete, we must show that it is as “hard" as a problem $B$ that is known to be NP-hard (which therefore itself is as “hard" as any problem in NP). What this means is that if we can solve $A$ in polynomial time, we should be able to solve $B$ in polynomial time as well. So, we create an algorithm that translates an instance $I_B$ for $B$ into an instance $I_A$ for $A$ in polynomial time, such that $I_A$ is a yes-instance for $A$ only if $I_B$ is a yes-instance for $B$. That way, if we can solve $A$ in polynomial time, we can translate any instance for $B$ to an instance for $A$, and solve the problem $A$, all in polynomial time, giving us a polynomial time algorithm for $B$.

If you manage to prove that a problem is NP-complete, you know that it will be unlikely that you find a polynomial time algorithm to solve that problem. If such an algorithm would exist, than it immediately implies that there exist polynomial time algorithms for all NP-complete problems. Since there have been a lot of researchers studying NP-completeness over the years, your algorithm would probably have to use some key insight that they have all missed, and your chances of having such an insight are small.

However, even if we have shown a problem to be NP-hard, we still want to find an efficient algorithm to solve that problem. This is where the complexity class FPT steps in. If we assume that our problem instance comes with some kind of parameter $k$, which could for instance denote the size of the solution, or some information about the problem instance, we can use this parameter to find an efficient algorithm. An algorithm is said to be in FPT if it's running time is $O(f(k)n^c)$ for some function $f$, and some constant $c$. The beauty of an FPT algorithm lies in the fact that for constant $k$, the algorithm runs in polynomial time. FPT algorithms are therefore useful if we know that $k$ will always be relatively small.

One popular strategy in developing an FPT algorithm is *kernelization*. Within kernelization, we define reduction rules to preprocess our problem instance in polynomial time. This usually means that we look at our problem instance, and find parts that we know won't be used in an optimal solution, or parts that need to be used in an optimal solution. This tells us either that we can remove said part from the problem instance, or start forming an optimal solution. For instance, if our problem is on graphs, we might know that independent vertices won't play a role and that we can remove those. Or, we might know that vertices of very high degree require special attention. After defining the reduction rules, we can apply them on a problem instance. If we take a problem instance $(I, k)$, and iteratively apply our reduction rules, we end up with a new problem instance $(I', k')$ where $k' \leq k$, such that $(I', k')$ is a yes-instance if and only if $(I, k)$ is a yes-instance.

Now, we look at our new instance $(I', k')$, and we know a lot about the structure of our instance, because we know we can no longer apply our reduction rules. At this point, we take our knowledge of what $k'$ denotes about our problem instance, and try to show that if $(I', k')$ is a yes-instance, then by the structure of $I'$, the size of $I'$ must be bounded by some function $g(k')$, and therefore also by $g(k)$. As our problem instance is now of size bounded by $g(k)$, we can use a naive approach to solve the problem instance in time exponential in its size, which means the time is bounded by some function of $k$. As the preprocessing was done in polynomial time, this will give us an FPT algorithm. For some examples of kernelization, we recommend Chapter 2 of [10].

## 2.2 Basic Definitions

In this section we will state some definitions used throughout this thesis. Let $G = (V, E)$ be a graph, with vertex set $V$ and edge set $E$. We also denote the vertex set of $G$ by $V(G)$, and the edge set of $G$ by $E(G)$. An edge from a vertex to itself is called a *loop*. Note that loops are not to be confused with cycles. We say that a graph has *parallel edges* if an edge appears multiple times. In this thesis we make the distinction between two types of graphs:

**Definition.** A graph $G = (V, E)$ is called a *simple graph*, if it contains no loops, and no parallel edges.

**Definition.** A graph $G = (V, E)$ is called a *unigraph*, if it contains no parallel edges. Note that unigraphs are allowed to have at most one loop per vertex.

By definition, every simple graph is a unigraph. As far as our knowledge goes, there is no formal name for graphs with no parallel edges that are allowed to have loops. That is why we introduce unigraphs as a name for this type of graphs. Furthermore, we define the following special type of unigraphs:

**Definition.** A unigraph $G = (V, E)$ is called a *reflexive unigraph* if for all $v \in V$, we have $\{v, v\} \in E$.

Recall the following definitions from graph-theory.

**Definition.** Given a graph $G = (V, E)$ we define a graph $G' = (V', E')$ to be an induced subgraph of $G$ if $V' \subseteq V$ and $E' = \{\{u, v\} : \forall u, v \in V'\} \cap E$. We say that $G'$ is the subgraph of $G$ induced on $V'$. We notate this as $G' = G[V']$.

**Definition.** Given a graph $G = (V, E)$, a vertex $v \in V$, and a vertex subset $X \subseteq V$. We define the *exclusive neighbourhood of* $v$, denoted by $N_G(v)$, as $N_G(v) = \{u \mid$

$u \in V$ and $\{u, v\} \in E\}$. We define the *inclusive neighbourhood of* $v$, denoted by $N_G[v]$, as $N_G[v] = N_G(v) \cup \{v\}$. We define the *inclusive neighbourhood of* $X$, denoted by $N_G[X] = \bigcup_{v \in X} N_G[v]$. Lastly, we define the *exclusive neighbourhood of* $X$, denoted by $N_G(X)$, as $N_G(X) = N_G[X] \setminus X$. Note that if vertex $v$ has a loop, then $N_G(v) = N_G[v]$.

**Definition.** A graph family $\mathcal{F}$ is a subset of all possible graphs.

**Definition.** Given a graph $H$, we define the $H$-free graphs as a graph family $\mathcal{F}$ such that $G \in \mathcal{F}$ if and only if there exists no $S \subseteq V(G)$ such that $G[S] = H$. i.e. $\mathcal{F}$ consists of all graphs that do not contain $H$ as an induced sub-graph.

We will also use the following concepts:

**Definition.** Give two sets $A$ and $B$, we define the symmetric difference of $A$ and $B$, denoted by $A \Delta B$, as

$$A \Delta B = (A \setminus B) \cup (B \setminus A)$$

**Definition.** Given an $n \times m$ matrix $M$, we define the *rank* of $M$ over a field $\mathbb{F}$ as the dimension of the linear subspace of $\mathbb{F}^n$ (or $\mathbb{F}^m$) that is spanned by the rows (or columns) of $M$.

Note that we already state that there is no distinction between row rank and column rank.

**Definition.** The *Galois Field* consisting of two elements (denoted by $GF[2]$) is the finite field on two elements. We will denote these two elements by $0$ and $1$. $GF[2]$ can also be defined as $\mathbb{Z}/2\mathbb{Z}$.

Note that the rank of a matrix $M$ depends on the field we take the rank over. As an example consider the following matrix:

$$M = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

In this case, we find that the rank over the reals is equal to $3$, as the three rows are linearly independent over $\mathbb{R}$. However, if we instead compute the rank over $GF[2]$, we find that

$$\begin{pmatrix} 0 & 1 & 1 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 \end{pmatrix}$$

Meaning that the rank over $GF[2]$ is equal to $2$. Within this project, we are exclusively interested in the rank over $GF[2]$, as all the matrices we discuss will be defined over $GF[2]$. Therefore, when we talk about rank, the reader can assume that we mean rank over $GF[2]$, unless we specify that we are working over some other field.

**Definition.** Given an $n \times m$ matrix $M$ and $I \subseteq \{1, \ldots, \min\{n, m\}\}$, we define the *principal submatrix on* $I$ as the submatrix of $M$ on the rows and columns with indices in $I$.

**Definition.** Let $\pi$ be a permutation of $n$ elements, thus $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ with $\pi$ bijective. Let $e_i$ denote the $i^{th}$ unit row vector. Then we define $P_\pi$, the *permutation matrix corresponding to* $\pi$, as

$$P_\pi = \begin{pmatrix} e_{\pi(1)} \\ e_{\pi(2)} \\ \vdots \\ e_{\pi(n)} \end{pmatrix}.$$

If we state that $P$ is a *permutation matrix*, than we mean that there exists some permutation $\pi$ such that $P = P_\pi$.

## 2.3 Rank-Width

Rank-width is a notion that proved itself to be valuable during the course of this project. In order to define rank-width, we first need to define a couple of other concepts. The following 4 definitions are taken from Oum [24].

**Definition** (Oum [24])**.** For a graph $G$ and a subset $X \subseteq V(G)$ we define the *cut-rank function* $\rho_G(X)$ to be the rank of a $|X| \times |V(G) - X|$, $GF[2]$ matrix $A_X$ where the entry of $A_X$ on the $i^{th}$ row and the $j^{th}$ column is 1 if and only if the $i^{th}$ vertex in $X$ is adjacent to the $j^{th}$ vertex in $V(G) - X$.

A tree in which every node has degree 1 or 3 is called *subcubic*.

**Definition** (Oum [24])**.** A *rank-decomposition* of a graph $G$ (containing at least two vertices) is a pair $(T, L)$ of a subcubic tree $T$ and a bijection $L$ from $V(G)$ to the set of all leaves of $T$.

**Definition** (Oum [24])**.** Given a graph $G$ and a rank-decomposition $(T, L)$ of $G$. For each edge $e$ of $T$, $T - e$ induces a partition $(A_e, B_e)$ of the leaves of $T$. We say that the *width* of $e$ is $\rho_G(L^{-1}(A_e))$. The *width* of $(T, L)$ is the maximum width of the edges of $T$.

**Definition** (Oum [24])**.** The *rank-width* of a graph $G$, denoted by $rw(G)$, is the minimum width over all rank-decompositions of $G$.

Note that if $G$ has less than two vertices, it does not have a rank-decomposition. In this case, we state that $rw(G) = 0$.

## 2.4 Clique-Width

In this section we define clique-width, introduced by Courcelle and Olariu [9]. This concept has proven to be valuable for this project. However, we will skip the proofs for some of the cited theorems, as they are relatively complicated, require further knowledge, and do not provide insight into our problem. Furthermore, since we exclude these proofs, an understanding of clique-width is not essential to understand this paper. However, we did feel it was necessary to include a definition, and state the theorem that we use to prove our problems are FPT.

We will limit our definition to undirected graphs. Let $\mathcal{C}$ be a countable set of labels. A *labelled* graph is a pair $(G, \gamma)$ where $\gamma$ maps $V(G)$ into $\mathcal{C}$. We say that $G$ is *$k$-labelled* if $\gamma$ is a vertex mapping from $V(G)$ to $\{1, \ldots, k\}$. We need the following definitions and operations:

1. For $a \in \mathcal{C}$, let $\cdot_a$ denote an isolated vertex labelled $a$.

2. For $a, b \in \mathcal{C}$, define the operator $\eta_{a,b}$ such that

$$\eta_{a,b}(G, \gamma) = (G', \gamma),$$

   where we define $G'$ such that $V(G') = V(G)$ and

$$E(G') = E(G) \cup \{\{u, v\} : u, v \in V(G), \gamma(u) = a, \gamma(v) = b\}.$$

   In other words, this operator creates all edges between vertices with label $a$ and vertices of label $b$.

3. For $a, b \in \mathcal{C}$, we define the operator $\rho_{a \to b}$ such that

$$\rho_{a \to b}(G, \gamma) = (G, \gamma')$$

   Where

$$\gamma'(v) = \begin{cases} b & \text{if } \gamma(v) = a, \\ \gamma(v) & \text{otherwise.} \end{cases}$$

   In other words, $\rho_{a \to b}$ changes the label of vertices labelled $a$ from $a$ to $b$.

4. Lastly, we use $\uplus$ to denote the disjoint union. Note that traditionally $\oplus$ is used to denote the disjoint union. However, as we reserve $\oplus$ to denote partial complements, we opted for this notation.

We can combine these symbols into a well-formed expression, which we call a $k$-expression (where $k$ is the amount of different labels used). A $k$-expression defines a $k$-labelled graph. This allows us to define clique width.

**Definition.** Given a graph $G$, the clique-width of $G$ (denoted by $\mathrm{cwd}(G)$) is equal to the minimal $k$ such that there exists a $k$-labelling $\gamma$ for which $(G, \gamma)$ can be formed by a $k$-expression.

Let us give a couple of examples. The complete simple graph on $4$ vertices, denoted by $K_4$, has clique-width $2$, as the version where every vertex has label $a$ can be written as:

$$\rho_{b\to a}(\eta_{a,b}(\cdot_b \uplus \rho_{b\to a}(\eta_{a,b}(\cdot_b \uplus \rho_{b\to a}(\eta_{a,b}(\cdot_b \uplus \cdot_a))))))$$

This approach can be extended to any complete simple graph $K_n$. Furthermore, if $G$ is a cluster graph, i.e. if every connected component of $G$ forms a clique, we know $\mathrm{cwd}(G) \leq 2$, as this graph is just the disjoint union of multiple complete simple graphs.

Now consider the cycle graph on $6$ vertices, denoted by $C_6$, and observe that if we assign each vertex label $a$, it can be expressed as:

$$\rho_{b\to a}(\rho_{c\to a}(\eta_{b,c}(\cdot_c \uplus \rho_{c\to a}(\eta_{a,c}(\cdot_c \uplus (\eta_{a,b}(\cdot_a \uplus \cdot_b) \uplus \eta_{a,b}(\cdot_a \uplus \cdot_b))))))))$$

## 2.5 Monadic Second Order Logic

In Section 3.2 we will use $\mathrm{MSO}_1$ (Monadic Second Order) logic on simple graphs to define our problems. Therefore, we will use this section to explain the concept of $\mathrm{MSO}_1$ logic on simple graphs in a way that is based on [10], without going into full detail. For further explanation we refer back to this book, though there probably exist more thorough sources. Within $\mathrm{MSO}_1$ logic on simple graphs, we can create logical formulas to verify a certain property of the graph.

Within $\mathrm{MSO}_1$ we are allowed to use three types of variables, namely single vertices, single edges, subsets of vertices. We can test interaction between these variables using the following operators:

1. $\in$; to test membership. Is this vertex part of this vertex subset?
2. $\mathbf{inc}(u, e)$; to test if edge $e$ is incident to a vertex $u$.
3. $=, \neq$; to test if two variables are equal, or not.

4. $\neg, \wedge, \vee, \Rightarrow$; the standard Boolean operators (negation, conjunction, disjunction, implication).

Finally, we can use quantifiers, which can be seen as counterparts of *loops* in standard programming languages. There are two types of quantifiers; the *universal quantifier* ($\forall$), and the *existential quantifier* ($\exists$). To explain how these work, suppose $\psi$ is a subformula that uses a single vertex variable $v$. If we then write $\forall_{v \in V} \psi$, that statement can be interpreted as "For every vertex $v$ in the graph, $\psi$ holds". On the other hand, $\exists_{v \in V} \psi$ can be interpreted as "There exists a vertex $v$ in the graph for which $\psi$ holds". In $\text{MSO}_1$ logic, quantification is not allowed over any type of variable. Specifically, we can not quantify over edge subsets. This is the crucial distinction between $\text{MSO}_1$ logic and the broader $\text{MSO}_2$ logic.

Let us give two examples of $\text{MSO}_1$ logic. The first will check if two vertices are adjacent, and the second will check if a set of elements are mutually disjoint.

$$\mathbf{adj}(u, v) = [\exists_{e \in E}(\mathbf{inc}(u, e) \wedge \mathbf{inc}(v, e))]$$

$$\begin{aligned} \textsc{MutDis}(i_1, \ldots, i_l) = [(&i_1 \neq i_2 \wedge i_1 \neq i_3 \wedge \ldots \wedge i_1 \neq i_l \wedge \\ &i_2 \neq i_3 \wedge i_2 \neq i_4 \wedge \ldots \wedge i_2 \neq i_l \wedge \\ &\vdots \\ &i_{l-1} \neq i_l)] \end{aligned}$$

# 3 FPT by Bounded Rank-Width

An important result by Fomin et al. [14] used a connection between partial complements and clique-width to prove the following result:

**Theorem 3.1.** *Let $\mathcal{F}$ be a graph family of bounded clique-width that is expressible in MSO$_1$ logic. Then, for simple graphs $G$, we can decide whether there exists $S \subseteq V(G)$ such that $G \oplus S \in \mathcal{F}$ in polynomial time.*

Within this chapter, we will extend this result to our setting. Specifically, we will show that $PC_{\mathcal{F}}(G, k)$ is FPT parametrized by $k$ for graph family's $\mathcal{F}$ of bounded rank-width for which $PC_{\mathcal{F}}(G, k)$ is expressible in MSO$_1$ logic. We will achieve this by using existing results on MSO$_1$ logic, clique-width and rank-width, and finding a new result on the relation between rank-width and partial complements. Note that it was already known that $\mathrm{cwd}(G \oplus S) \leq 3\mathrm{cwd}(G)$ [14]. However, we are able to show that

$$rw(G) - 1 \leq rw(G') \leq rw(G) + 1.$$

This has some advantages to the previously known result. Firstly, combined with Theorem 3.4, this will lead to a lower bound on the clique-width of a graph created by taking an unconnected set of vertices and taking multiple partial complements. Secondly, there is no known polynomial time algorithm to decide whether a graph has clique-width at most $k$ [16, 24], whereas deciding whether or not a graph has rank-width at most $k$ can be done in polynomial time [23].

If the reader is unfamiliar with MSO$_1$ logic, clique-width and/or rank-width, there is a brief description of these concepts in the Chapter 2. Note that we exclude proofs for the cited results we use, as they would require an understanding of these concepts that is more in-depth than is necessary for understanding the results in this thesis.

This chapter is divided into two sections. In the first section we will prove the extension of Theorem 3.1. In the second section, we will give examples of graph families that satisfy the conditions given in this extension, to show that this result is indeed useful. Namely, we show that a family of $H$-free graphs satisfies the conditions if and only if $H$ is an induced subgraph of $P_4$.

## 3.1 Proving FPT

We start this section by stating the theorem we want to prove. After this, we will cite the results that we need. Then, we will prove that taking a partial complement has a bounded effect on the rank-width of a graph, i.e. the change in rank-width will never be greater than some constant. Then we combine all of these results to prove our main theorem of this chapter:

**Theorem 3.2.** *Let $\mathcal{F}$ be a family of graphs such that there exists a constant $c$ for which $G \in \mathcal{F}$ implies rw$(G) \leq c$, and the decision problem $PC_{\mathcal{F}}(G, k)$ can be expressed with an MSO$_1$ formula. Then $PC_{\mathcal{F}}(G, k)$ is FPT parametrized by $k$.*

In order to prove this, we will need to use some existing results. A key part in this proof will be the following theorem by Courcelle, Makowsky, and Rotics [8], which states:

**Theorem 3.3** (Courcelle, Makowsky and Rotics)**.** *Every graph decision problem specified by a formula of MSO$_1$ logic is fixed parameter tractable with regards to clique-width as a parameter, if we are given a $k$-expression of the graph.*

We will exclude this proof, as it uses a lot of knowledge that is not required to understand the rest of this thesis. Please note that the original phrasing of this theorem differs from our phrasing, which more closely resembles the statement as phrased by Oum [24]. In order to achieve a proof for Theorem 3.2, we will need to show a relation between $k$ and the clique-width of a graph. To achieve this, we use the following theorem, due to Oum and Seymour [25], which shows a relation between rank-width and clique-width. For a proof of this theorem we redirect you to the original paper, where it is denoted as proposition 6.3.

**Theorem 3.4** (Oum and Seymour [25])**.** *For any simple graph $G$, it holds that*

$$\text{rw}(G) \leq \text{cwd}(G) \leq 2^{\text{rw}(G)+1} - 1$$

Before we bring in our own results, we will require one more external result, stating that we can check whether the rank-width of a graph is at most $k$ in time polynomial in the graph size (for fixed $k$). We will again exclude the proof.

**Lemma 3.5** (Oum and Seymour [25])**.** *Let $k$ be a fixed positive integer. There exists an $O(n^9 \log(n))$-time algorithm that confirms whether or not an $n$-vertex simple graph $G$ has rank-width at most $k$.*

We have now cited all results that we require for the proof of Theorem 3.2. Let us turn to our original results. For starters, we will prove a bound on the difference a partial complement makes on the rank-width of a simple graph:

**Theorem 3.6.** *Given simple graph $G = (V, E)$, and $S \subseteq V$. If a graph $G'$ is defined as $G' = G \oplus S$, then it holds that*

$$rw(G) - 1 \leq rw(G') \leq rw(G) + 1.$$

*Proof.* Let $(T, L)$ be a width minimizing rank decomposition for $G$, i.e. such that the width of $(T, L)$ corresponds to the rank-width of $G$. Since $V(G') = V(G)$, we know that $(T, L)$ also functions as a rank decomposition for $G'$. Consider an edge $e$ in $T$, and let $(A_e, B_e)$ be the partition of the vertices it produces. Let $M$ and $M'$ denote the matrices corresponding to $\rho_G(L^{-1}(A_e))$ and $\rho_{G'}(L^{-1}(A_e))$ respectively. If $S \subseteq L^{-1}(A_e)$ or $S \subseteq L^{-1}(B_e))$, then $M = M'$. Else, let $X$ be a $|A_e| \times |B_e|$ $GF[2]$ matrix where $X_{i,j} = 1$ if and only if the $i^{th}$ element of $A_e$ and the $j^{th}$ element of $B_e$ are both mapped by $L^{-1}$ to an element in $S$. By the definition of the partial complement, it must than hold that $M' = M + X$. Since $\mathrm{rk}\,(X) = 1$, we know that $\mathrm{rk}\,(M') \leq \mathrm{rk}\,(M) + 1$. As this holds for any edge $e$, we can conclude that $rw(G') \leq rw(G) + 1$, as we have found a rank decomposition for $G'$ with width at most $rw(G) + 1$.

Now note that by the symmetry of the partial complement, we have that $G = G' \oplus S$. Following the same reasoning, this tells us the $rw(G) \leq rw(G') + 1$. Combining this with the previous result gives us the desired statement. ■

We want to give an example to prove that the bounds given in Theorem 3.6 are tight. Consider a $C_5$ graph, i.e. the cycle on 5 vertices. We know that $rw(C_5) = 2$, as for any $X \subseteq V(C_5)$ with $|X| = 2$ we see that $\rho_{C_5}(X) = 2$. If we take a partial complement on two adjacent vertices, we will remove one edge and create a $P_5$ graph, which clearly has rank-width 1. Giving us an example where $rw(G') = rw(G) - 1$. Of course, if we would reverse this operation we would find that turning $P_5$ into $C_5$ gives us an example where $rw(G') = rw(G) + 1$. Lastly, if we start with $C_5$, and take the partial complement on all vertices, we will again have a $C_5$ graph, giving us an example where $rw(G') = rw(G)$. Thus, for all three possible outcomes of the rank-width after taking a partial complement, we can find an example where it occurs. Let us now consider the following corollary, which follows from Theorem 3.6.

**Corollary 3.6.1.** *Let $\mathcal{F}$ be a graph family with rank-width bounded by constant $c$, i.e. there exists some $c$ such that $rw(G) \leq c$ for all $G \in \mathcal{F}$. If $PC_{\mathcal{F}}(G, k)$ is a yes-instance, then it holds that $rw(G) \leq c + k$.*

*Proof.* We know that the graph family $\mathcal{F}$ has rank-width bounded by $c$. By Theorem 3.6, we know that every partial complement can decrease the rank-width by at most 1. Therefore, if $rw(G) > c + k$, we know that we cannot get a graph of rank-width $c$ by taking $k$ partial complements on $G$, which therefore rules out the possibility that we can turn $G$ into a graph $G' \in \mathcal{F}$ by taking at most $k$ partial complements. ■

This gives us sufficient information to prove Theorem 3.2:

*Proof of Theorem 3.2.* Let $G$ be an input graph for $PC_{\mathcal{F}}(G, k)$, where $\mathcal{F}$ satisfies the conditions we gave. By Corollary 3.6.1, we know that if $rw(G)$ is larger than $c + k$, $G$ must be a no-instance. So, in determining whether or not $PC_{\mathcal{F}}(G, k)$ is a yes-instance, the first step will be to determine the rank-width of $G$. By Lemma 3.5, this can be done in time FPT with regards to $k$. After this step, if $PC_{\mathcal{F}}(G, k)$ is still undecided, we know that $rw(G) \leq k + c$.

Applying Theorem 3.4, we know that this implies that $\mathrm{cwd}(G) \leq 2^{k+c+1}-1$. We know that the decision problem $PC_{\mathcal{F}}(G, k)$ is expressible in $MSO_1$ logic. By Theorem 3.3, we know that this means that $PC_{\mathcal{F}}(G, k)$ is solvable in time bounded by $f(\mathrm{cwd}(G)) \cdot (|V(G)| + |E(G)|)^{c_2}$, for some constant $c_2$. As we have bounded the clique-width of $G$ by a function of $k$, this implies that $PC_{\mathcal{F}}(G, k)$ is solvable in time bounded by $f'(k) \cdot (|V(G)| + |E(G)|)^{c_2}$, for some constant $c_2$. In other words, $PC_{\mathcal{F}}(G, k)$ is FPT with regards to parameter $k$. ■

The proof of Theorem 3.3 given in [8] is constructive, meaning it directly implies an algorithm to solve the decision problem. As the proof of Theorem 3.2 is built on Theorem 3.3, this algorithm extends to our case. So, we now have an $O(f(k)(|V(G)| + |E(G)|)^{c_2})$ time algorithm to solve $PC_{\mathcal{F}}(G, k)$ for some graph families $\mathcal{F}$. Unfortunately, the bound used for $f(k)$ in the proof by Courcelle, Makowsky, and Rotics is gigantic. They require a $k'$-expression for $G$. By [25], we know that we can find a $k'$-expression for $G$ where $k' \leq (2^{3(rw(G))+2} - 1)$. As we know $G$ has bounded rank-width, we can say that $k' \leq (2^{3(k+c)+2} - 1)$ for some constant $c$, depending on the choice of $\mathcal{F}$. The bound given for $f(k)$ is then the amount of $\mathrm{MSO}_1$ formulas on $2^{3(k+c)+2} - 1$ labels, with a number of quantifiers at most the number of quantifiers used by the $\mathrm{MSO}_1$ formula that expresses $PC_{\mathcal{F}}(G, k)$. In general, the dependence of the running time of an application of the algorithm from Theorem 3.3 on the formula describing the problem is a tower of exponentials of unbounded height [20]. When considering a particular problem, there might be a nicer bound achievable, but for our problem this is not obvious.

## 3.2 Examples of Graph Families Satisfying the Constraints

Now that the proof of Theorem 3.2 is complete, let us study some of the graph families that satisfy the constraints. We are specifically interested in families of $H$-free simple graphs, for $H$ some simple graph. As Theorem 3.2 requires graph families of bounded rank-width, which also implies bounded clique-width, the following result is noteworthy:

**Theorem 3.7** (Dabrowski and Paulusma [11])**.** *Let $H$ be a simple graph. The family of $H$-free simple graphs has bounded clique-width if and only if $H$ is an induced subgraph of $P_4$.*

The induced subgraphs of $P_4$ are $P_1, P_2, \overline{P_2}, P_3, \overline{P_3}$, and $P_4$. A $P_1$-free graph cannot have vertices. As partial complements cannot add or remove vertices, this graph family is clearly closed under partial complements. Therefore, $PC_{\mathcal{F}}(G)$ where $\mathcal{F}$ is the family of $P_1$-free simple graphs is trivial. The other induced subgraphs of $P_4$ are more interesting.

In this section, we will show that the family of $P_2$-free simple graphs, the family of $\overline{P_2}$-free simple graphs, the family of $P_3$-free simple graphs, the family of $\overline{P_3}$-free simple graphs, and the family of $P_4$-free simple graphs all satisfy our constraints. We know by Theorem 3.7 and Theorem 3.4 that these families have bounded rank-width. So, all we have to show is that the decision problems $PC_{\mathcal{F}}(G, k)$ are expressible in $\text{MSO}_1$ logic for these graphs families $\mathcal{F}$. To achieve this we will start by expressing a membership test in $\text{MSO}_1$ logic, after which we will show how to account for partial complements. Recall the following two formulas, introduced in Section 2.5:

$$\begin{aligned}\textsc{MutDis}(i_1, \ldots, i_l) = [&(i_1 \neq i_2 \wedge i_1 \neq i_3 \wedge \ldots \wedge i_1 \neq i_l \wedge \\ &i_2 \neq i_3 \wedge i_2 \neq i_4 \wedge \ldots \wedge i_2 \neq i_l \wedge \\ &\vdots \\ &i_{l-1} \neq i_l)]\end{aligned}$$

$$\textbf{adj}(u, v) = [\exists_{e \in E}(\textbf{inc}(u, e) \wedge \textbf{inc}(v, e))]$$

Since our graph families are centered around induced subgraphs, let us express whether some vertices express a forbidden subgraph $H$ within $\text{MSO}_1$ logic. Then, we can test membership in our $H$-free graph families by quantifying over all vertices that could possible induce $H$, and checking that they don't induce $H$. We know that

we can express whether vertices $u, v$ induce $P_2$ or $\overline{P_2}$ by $\mathbf{adj}(u, v)$ and $\neg\mathbf{adj}(u, v)$. We can also express this for $P_3, \overline{P_3}$, and $P_4$, where the bracket colour is used simply for emphasis:

$$\begin{aligned}\textsc{Induce-}P_3(u,v,w) =&[((\mathbf{adj}(u,v)) \Rightarrow ((\mathbf{adj}(u,w) \wedge \neg\mathbf{adj}(v,w)) \vee (\neg\mathbf{adj}(u,w) \wedge \mathbf{adj}(v,w))))\\ &\wedge ((\neg\mathbf{adj}(u,v)) \Rightarrow (\mathbf{adj}(u,w) \wedge \mathbf{adj}(v,w)))]\end{aligned}$$

$$\begin{aligned}\textsc{Induce-}\overline{P_3}(u,v,w) =&[((\neg\mathbf{adj}(u,v)) \Rightarrow ((\mathbf{adj}(u,w) \wedge \neg\mathbf{adj}(v,w)) \vee (\neg\mathbf{adj}(u,w) \wedge \mathbf{adj}(v,w))))\\ &\wedge ((\mathbf{adj}(u,v)) \Rightarrow (\neg\mathbf{adj}(u,w) \wedge \neg\mathbf{adj}(v,w)))]\end{aligned}$$

$$\begin{aligned}\textsc{Induce-}P_4(u,v,w,z) =&[(\textsc{Induce-}P_3(u,v,w) \Rightarrow\\ &(\textsc{Induce-}P_3(u,v,z) \wedge \neg\textsc{Induce-}P_3(u,w,z) \wedge \neg\textsc{Induce-}P_3(v,w,z))\\ &\vee (\neg\textsc{Induce-}P_3(u,v,z) \wedge \textsc{Induce-}P_3(u,w,z) \wedge \neg\textsc{Induce-}P_3(v,w,z))\\ &\vee (\neg\textsc{Induce-}P_3(u,v,z) \wedge \neg\textsc{Induce-}P_3(u,w,z) \wedge \textsc{Induce-}P_3(v,w,z)))\\ &\wedge (\textsc{Induce-}\overline{P_3}(u,v,w) \wedge \mathbf{adj}(u,v) \Rightarrow (\mathbf{adj}(w,z) \wedge \textsc{Induce-}P_3(u,v,z)))\\ &\wedge (\textsc{Induce-}\overline{P_3}(u,v,w) \wedge \mathbf{adj}(u,w) \Rightarrow (\mathbf{adj}(v,z) \wedge \textsc{Induce-}P_3(u,w,z)))\\ &\wedge (\textsc{Induce-}\overline{P_3}(u,v,w) \wedge \mathbf{adj}(v,w) \Rightarrow (\mathbf{adj}(u,z) \wedge \textsc{Induce-}P_3(v,w,z)))]\end{aligned}$$

We now need to incorporate partial complements. In a partial complement, we can pick subsets of the vertices to complement all the edges within this subset. Because of this, it is interesting to know how many times a vertex pair $u, v$ is contained in such a subset. Therefore we introduce the even parity function, that, given vertex subsets $X_1, \ldots, X_l$ and vertex pair $u, v$, checks if the amount of subsets that contain both $u$ and $v$ is even. This is useful, because if this is even, then the edge/non-edge between $u$ and $v$ will be preserved after taking partial complements on $X_1, \ldots, X_l$, whereas if they are contained in an odd number of subsets, the edge/non-edge will be flipped after taking the partial complements.

$$\begin{aligned}\textsc{EvPa}(X_1,\ldots,X_l,u,v) =&[(\exists_{k\in\{0,\ldots,\lfloor\frac{l}{2}\rfloor\}}\exists_{j_1\ldots,j_{2k}\in\{1,\ldots,l\}}\forall_{i\in\{1,\ldots,l\}\setminus\{j_1,\ldots,j_{2k}\}}\forall_{j\in\{j_1,\ldots,j_{2k}\}}\\ &\textsc{MutDis}(j_1,\ldots,j_{2k}) \wedge (u,v \in X_j) \wedge (u \notin X_i \vee v \notin X_i))]\end{aligned}$$

Please note that the quantification over the indices is actually shorthand for a case distinction of all the different ways to divide $X_1, \ldots, X_l$ into two parts such that one part has an even number of subsets, and then quantifying over all these vertex

subsets accordingly, meaning our expressing still fits the rules of $\text{MSO}_1$ logic. The expression of $\text{EvPa}(X_1, \ldots, X_l, u, v)$ allows us to write a formula that checks whether two vertices will be adjacent after taking the partial complements.

$$\begin{aligned}\mathbf{adj}^l(X_1, \ldots, X_l, u, v) = [&(\mathbf{adj}(u, v) \wedge \text{EvPa}(X_1, \ldots, X_l, u, v)) \\ &\vee (\neg\mathbf{adj}(u, v) \wedge \neg\text{EvPa}(X_1, \ldots, X_l, u, v))]\end{aligned}$$

This adjacency after partial complements formula allows us to create a function that verifies whether a set of vertices will induce $P_3$, $\overline{P_3}$, or $P_4$ after a set of partial complements, based on the formulas that check whether a set of vertices induce $P_3$, $\overline{P_3}$ or $P_4$.

$$\begin{aligned}&\text{Induce-}P_3^l(X_1, \ldots, X_l, u, v, w) = \\ &\quad\Big[((\mathbf{adj}^l(X_1, \ldots, X_l, u, v)) \Rightarrow ((\mathbf{adj}^l(X_1, \ldots, X_l, u, w) \wedge \neg\mathbf{adj}^l(X_1, \ldots, X_l, v, w)) \\ &\qquad\vee(\neg\mathbf{adj}^l(X_1, \ldots, X_l, u, w) \wedge \mathbf{adj}^l(X_1, \ldots, X_l, v, w)))) \\ &\quad\wedge((\neg\mathbf{adj}^l(X_1, \ldots, X_l, u, v)) \Rightarrow (\mathbf{adj}^l(X_1, \ldots, X_l, u, w) \wedge \mathbf{adj}^l(X_1, \ldots, X_l, v, w)))\Big]\end{aligned}$$

$$\begin{aligned}&\text{Induce-}\overline{P_3}^l(X_1, \ldots, X_l, u, v, w) = \\ &\quad\Big[((\neg\mathbf{adj}^l(X_1, \ldots, X_l, u, v)) \Rightarrow ((\mathbf{adj}^l(X_1, \ldots, X_l, u, w) \wedge \neg\mathbf{adj}^l(X_1, \ldots, X_l, v, w)) \\ &\qquad\vee(\neg\mathbf{adj}^l(X_1, \ldots, X_l, u, w) \wedge \mathbf{adj}^l(X_1, \ldots, X_l, v, w)))) \\ &\quad\wedge((\mathbf{adj}^l(X_1, \ldots, X_l, u, v)) \Rightarrow (\neg\mathbf{adj}^l(X_1, \ldots, X_l, u, w) \wedge \neg\mathbf{adj}^l(X_1, \ldots, X_l, v, w)))\Big]\end{aligned}$$

$$\begin{aligned}&\text{Induce-}P_4^l(X_1, \ldots, X_l, u, v, w, z) = \Big[((\text{Induce-}\overline{P_3}^l(X_1, \ldots, X_l, u, v, w)) \Rightarrow \\ &\quad((\text{Induce-}\overline{P_3}^l(X_1, \ldots, X_l, u, v, z) \wedge \neg\text{Induce-}\overline{P_3}^l(X_1, \ldots, X_l, u, w, z) \\ &\qquad\wedge\neg\text{Induce-}\overline{P_3}^l(X_1, \ldots, X_l, v, w, z)) \\ &\quad\vee(\neg\text{Induce-}\overline{P_3}^l(X_1, \ldots, X_l, u, v, z) \wedge \text{Induce-}\overline{P_3}^l(X_1, \ldots, X_l, u, w, z) \\ &\qquad\wedge\neg\text{Induce-}\overline{P_3}^l(X_1, \ldots, X_l, v, w, z)) \\ &\quad\vee(\neg\text{Induce-}\overline{P_3}^l(X_1, \ldots, X_l, u, v, z) \wedge \neg\text{Induce-}\overline{P_3}^l(X_1, \ldots, X_l, u, w, z) \\ &\qquad\wedge\text{Induce-}\overline{P_3}^l(X_1, \ldots, X_l, v, w, z)))) \\ &\quad\wedge((\text{Induce-}\overline{P_3}^l(X_1, \ldots, X_l, u, v, w) \wedge \mathbf{adj}^l(X_1, \ldots, X_l, u, v)) \Rightarrow \\ &\qquad(\mathbf{adj}^l(X_1, \ldots, X_l, w, z) \wedge \text{Induce-}\overline{P_3}^l(X_1, \ldots, X_l, u, v, z))) \\ &\quad\wedge((\text{Induce-}\overline{P_3}^l(X_1, \ldots, X_l, u, v, w) \wedge \mathbf{adj}^l(X_1, \ldots, X_l, u, w)) \Rightarrow \\ &\qquad(\mathbf{adj}^l(X_1, \ldots, X_l, v, z) \wedge \text{Induce-}\overline{P_3}^l(X_1, \ldots, X_l, u, w, z))) \\ &\quad\wedge((\text{Induce-}\overline{P_3}^l(X_1, \ldots, X_l, u, v, w) \wedge \mathbf{adj}^l(X_1, \ldots, X_l, v, w)) \Rightarrow \\ &\qquad(\mathbf{adj}^l(X_1, \ldots, X_l, u, z) \wedge \text{Induce-}\overline{P_3}^l(X_1, \ldots, X_l, v, w, z)))\Big]\end{aligned}$$

This allows us to express our problems in $\text{MSO}_1$ logic:

$$
\begin{aligned}
\text{NO}P_2\text{IN}(G,l) =& \exists_{X_1,\ldots,X_l \subseteq V(G)}[\forall_{u,v \in V(G)}(u \neq v) \Rightarrow \neg \mathbf{adj}^l(X_1,\ldots,X_l,u,v)] \\
\text{NO}\overline{P_2}\text{IN}(G,l) =& \exists_{X_1,\ldots,X_l \subseteq V(G)}[\forall_{u,v \in V(G)}(u \neq v) \Rightarrow \mathbf{adj}^l(X_1,\ldots,X_l,u,v)] \\
\text{NO}P_3\text{IN}(G,l) =& \exists_{X_1,\ldots,X_l \subseteq V(G)}[\forall_{u,v,w \in V(G)}\text{MUTDIS}(u,v,w) \Rightarrow \\
& \neg\text{INDUCE-}P_3^l(X_1,\ldots,X_l,u,v,w)] \\
\text{NO}\overline{P_3}\text{IN}(G,l) =& \exists_{X_1,\ldots,X_l \subseteq V(G)}[\forall_{u,v,w \in V(G)}\text{MUTDIS}(u,v,w) \Rightarrow \\
& \neg\text{INDUCE-}\overline{P_3}^l(X_1,\ldots,X_l,u,v,w)] \\
\text{NO}P_4\text{IN}(G,l) =& \exists_{X_1,\ldots,X_l \subseteq V(G)}[\forall_{u,v,w,z \in V(G)}\text{MUTDIS}(u,v,w,z) \Rightarrow \\
& \neg\text{INDUCE-}P_4^l(X_1,\ldots,X_l,u,v,w,z)]
\end{aligned}
$$

Thus, we can conclude that if $\mathcal{F}$ is the family of $H$-free simple graphs, where $H$ is an induced subgraph of $P_4$, $PC_{\mathcal{F}}(G,k)$ is FPT parametrized by $k$ as a result of Theorem 3.2.

# 4 Unigraphs and Symmetric Rank

In this chapter we look at a variation of our problem. In this variation, we consider unigraphs, i.e. graphs that are allowed to contain loops. Note that unigraphs is a name that we introduced in this thesis. Recall the definition of partial complements on unigraphs:

**Definition.** Given a unigraph $G = (V, E)$, let $S \subseteq V$ be a subset of vertices. The *partial complement on* $S$, denoted by $G\widehat{\oplus}S$, returns a unigraph $G' = (V, E')$ such that:

$$E' = E\Delta\{\{u, v\} : \forall u, v \in S\}$$

Alternatively, we could define $E'$ in the following manner.

$$\begin{aligned} \{u, v\} \in E' &\Leftrightarrow \{u, v\} \notin E && \forall u, v \in S \\ \{u, v\} \in E' &\Leftrightarrow \{u, v\} \in E && \forall u, v \text{ not both in } S \end{aligned}$$

Note that in this definition, $u$ and $v$ can be the same vertex. Thus, partial complements on unigraphs will add/remove loops at every vertex contained in the partial complement. The benefit of this operator compared to the original partial complement follows from the following reasoning:

Given a unigraph $G$ with corresponding adjacency matrix $A_G$. For $S \subseteq V(G)$, let $C_S$ be the $|V(G)| \times |V(G)|$ matrix defined as $C_S[i, j] = 1$ if and only if $i \in S$ and $j \in S$, and $0$ otherwise. Note that $i$ and $j$ do not need to be distinct. Consider the unigraph whose adjacency matrix is $A_G + C_S$, and observe that it is exactly equal to $G\widehat{\oplus}S$. For future reference, we express this property in the following observation:

**Observation 1.** *Given a unigraph $G$ with corresponding adjacency matrix $A_G$, and vertex subset $S \subseteq V(G)$. There exists a symmetric $GF[2]$ rank one matrix $C_S$ such that $A_G + C_S = A_{(G\widehat{\oplus}S)}$*

Observation 1 generates an interest into symmetric rank one matrices. We therefore introduce the following concepts, which will prove to be fruitful over the course of this chapter:

**Definition.** Given a symmetric matrix $A$ over $GF[2]$, we define a *symmetric rank decomposition* of $A$ to be a set of $GF[2]$ column vectors $u_1 \dots u_k$, such that

$$A = \sum_{i=1}^{k} u_i {u_i}^T.$$

**Definition.** Given a symmetric matrix $A$ over $GF[2]$, we define a *symmetric rank* of $A$, denoted by $\mathrm{rk}_{\mathbb{S}}(A)$, to be the size of the smallest symmetric rank decomposition of $A$.

Recall that to eliminate confusion, we write $\widehat{PC_{\mathcal{F}}}(G)$ to denote a problem instance in which loops are allowed. Note that as with the normal partial complement, $|\widehat{PC_{\mathcal{F}}}(G)|$ denotes the size of the minimal family of partial complement required to turn unigraph $G$ into a unigraph $G' \in \mathcal{F}$, and $\widehat{PC_{\mathcal{F}}}(G,k)$ denotes the decision variant of $\widehat{PC_{\mathcal{F}}}(G)$.

Furthermore, as the family of graphs we are aiming for is now allowed to have loops, we introduce the following notion:

**Definition.** Let $\mathcal{F}$ be a family of simple graphs. We call a family of unigraphs $\widehat{\mathcal{F}}$ a *loop family extension of* $\mathcal{F}$ if for every unigraph $\widehat{G} \in \widehat{\mathcal{F}}$ it holds that $(V(\widehat{G}), E(\widehat{G}) \setminus L) \in \mathcal{F}$, where $L = \{\{v,v\} \mid \forall v \in V(\widehat{G})\} \cap E(\widehat{G})$, and for every simple graph $G \in \mathcal{F}$ there exists $S \subseteq V(G)$ such that 'E$(V(G), E(G) \cup L') \in \widehat{\mathcal{F}}$, where $L' = \{\{v,v\} \mid \forall v \in S\}$.

This gives us enough vocabulary to look at specific graph families.

## 4.1 Edgeless Graphs

In this section, we focus on $\widehat{PC_{\mathcal{G}}}(G)$, where $\mathcal{G}$ is the family of edgeless graphs. Note that $\mathcal{G}$ is a loop family extension of itself, and that we made a conscious choice to not add loops to the graphs we aim to obtain. Since the adjacency matrix of an edgeless graph is always the all-zero matrix, the symmetric rank decomposition of an adjacency matrix is key for solving $\widehat{PC_{\mathcal{G}}}(G)$:

Let $G$ be a unigraph with corresponding adjacency matrix $A_G$, Let $u_1, \dots, u_k$ denote a symmetric rank decomposition of $A_G$ of minimal size, and let $S_1, \dots, S_k$ be vertex subsets such that $u_i$ is an indicator vector for membership in $S_i$. Observe now that $A_{G \widehat{\oplus} S_1} = A_G + u_1 u_1^T$. Furthermore, observe that this relation will continue to hold for additional $u_i$'s. This tells us that taking partial complements on $S_1, \dots, S_k$ will remove all the edges. Now, let $S_1, \dots, S_k$ be a solution to $\widehat{PC_{\mathcal{G}}}(G)$ (implying $k$ is

minimal), and define the $u_i$'s to be the indicator vectors for membership in the $S_i$. Observe that the same relations must hold, i.e. the $u_i$'s must form a symmetric rank decomposition. As our relation holds both ways, a minimal symmetric rank decomposition will produce a solution to $\widehat{PC_{\mathcal{G}}}(G)$, and a solution to $\widehat{PC_{\mathcal{G}}}(G)$ will produce a minimal symmetric rank decomposition. For future reference, we express this property in the following observation:

**Observation 2.** *Given a unigraph $G$ and corresponding adjacency matrix $A_G$. Any minimal symmetric rank decomposition of $A_G$ defines a solution to $\widehat{PC_{\mathcal{G}}}(G)$, and vice versa.*

Throughout this section, we will work towards a proof for the following theorem:

**Theorem 4.1.** *Let $A$ be a symmetric $GF[2]$ matrix, such that $A$ is not the all zero matrix. If $A$ has no $1$ on its diagonal, then $\mathrm{rk}_{\mathbb{S}}(A) = \mathrm{rk}(A) + 1$. Otherwise, $\mathrm{rk}_{\mathbb{S}}(A) = \mathrm{rk}(A)$.*

Before we discuss how we can prove this Theorem, let us translate this result back to the partial complement setting in the following corollary.

**Corollary 4.1.1.** *Let $G$ be a unigraph with corresponding adjacency matrix $A_G$, such that $E(G) \neq \emptyset$, and let $\mathcal{G}$ be the family of edgeless graphs. If $G$ contains no loops then $|\widehat{PC_{\mathcal{G}}}(G)| = \mathrm{rk}(A_G) + 1$. Else, $|\widehat{PC_{\mathcal{G}}}(G)| = \mathrm{rk}(A_G)$.*

*Proof.* Apply Theorem 4.1 on $A_G$ to find a symmetric rank decomposition of the correct size, and use Observation 2 to translate this to a solution for $\widehat{PC_{\mathcal{G}}}(G)$. ■

To achieve a proof for Theorem 4.1 we will first find an approximation algorithm whose output is at most one larger than the optimal value of $|\widehat{PC_{\mathcal{G}}}(G)|$. After this, we will study the relation between the symmetric rank of a matrix, and its regular rank. Specifically, we study when these two are not equal. Combining all these results will give us everything we need to prove Theorem 4.1.

### 4.1.1 Approximation Algorithm

We will find an approximation algorithm whose output is at most one larger than the optimal value of $|\widehat{PC_{\mathcal{G}}}(G)|$. To achieve this, we will first prove a lower bound for $|\widehat{PC_{\mathcal{G}}}(G)|$ in Lemma 4.2. Then, we will find an algorithm that, given symmetric matrix $A$, outputs a set of symmetric rank one matrices and a set of non-symmetric rank one matrices paired with their transpose, who together sum up to $A$. We will

prove correctness of this algorithm in Lemma 4.3. This algorithm will serve as a base for our final approximation algorithm. Lemma 4.4 then shows how we can use the output of our algorithm to find a symmetric rank decomposition. This then leads us to the desired approximation algorithm. The core concept of this approximation is then given in Theorem 4.5. But first, the lower bound for $|\widehat{PC_{\mathcal{G}}}(G)|$:

**Lemma 4.2.** *Let $A$ be a symmetric $GF[2]$ matrix. It holds that:*

$$\mathrm{rk}_{\mathbb{S}}(A) \geq \mathrm{rk}(A)$$

*Proof.* Let $u_1, \ldots, u_k$ be a symmetric rank decomposition of $A$ of minimal size. Then it holds that

$$A = \sum_{i=1}^{k} u_i {u_i}^T$$

Clearly:

$$\mathrm{rk}\left(\sum_{i=1}^{j+1} u_i {u_i}^T\right) \leq \mathrm{rk}\left(\sum_{i=1}^{j} u_i {u_i}^T\right) + 1 \qquad \forall j \in \{0, \ldots, k-1\}$$

Which implies that $\mathrm{rk}(A) \leq k = \mathrm{rk}_{\mathbb{S}}(A)$. ■

Let us give an example in which the inequality of Lemma 4.2 is strict. Let $A$ be the following matrix:

$$A = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

The rank of $A$ is equal to 2. However, there is no combination of 2 $3 \times 3$ symmetric rank one matrices that add up to $A$. Though this example is small enough to be checked by hand, the statement will also follow from Lemma 4.7.

We know that the symmetric rank of any matrix $A$ will be lower-bound by its regular rank. In Theorem 4.1, we will prove a more involved relation between the symmetric rank and the regular rank.

We know that for any symmetric matrix $A$, $\mathrm{rk}(A) \leq \mathrm{rk}_{\mathbb{S}}(A)$, but we still need an algorithm to find a symmetric rank decomposition. Based on an answer given in [15], we describe the following algorithm that takes a symmetric $GF[2]$ matrix $A$ as input, and outputs $u_1, \ldots, u_k$, $x_1, \ldots, x_l$, and $y_1, \ldots, y_l$, such that $k + 2l = \mathrm{rk}(A)$ and

$$A = \sum_{i=1}^{k} u_i {u_i}^T + \sum_{j=1}^{l} (x_i {y_i}^T + y_i {x_i}^T).$$

We will later use this output to create an approximation algorithm for an optimal symmetric rank decomposition. We name this algorithm *Sym_Rank_Base*$(A)$.

*Sym_Rank_Base*$(A)$:
Denote the number of columns in $A$ by $n$. The algorithm distinguishes a couple of cases:

1. In the case where $A$ is the all zero matrix, we know that an empty symmetric rank decomposition suffices, so that is what we return.

2. If $A$ has a non-zero diagonal entry, apply row and column swapping with a permutation matrix $P$ such that

$$PAP^T = \begin{pmatrix} 1 & v^T \\ v & A' \end{pmatrix} = \begin{pmatrix} 1 \\ v \end{pmatrix} \begin{pmatrix} 1 & v^T \end{pmatrix} + \begin{pmatrix} 0 & \vec{0}^T \\ \vec{0} & A' - vv^T \end{pmatrix}$$

   Where $\vec{0}$ denotes the all zero column vector. Recursing on our algorithm, we can find the solution $u'_1, \ldots, u'_{k-1}, x'_1, \ldots, x'_l, y'_1, \ldots, y'_l$ for $A' - vv^T$. This gives us the solution

$$\begin{aligned} u_k &= P^T \begin{pmatrix} 1 \\ v \end{pmatrix} P \\ u_i &= P^T \begin{pmatrix} 0 \\ u'_i \end{pmatrix} P \qquad \text{for } i \in \{1, \ldots, k-1\} \\ x_i &= P^T \begin{pmatrix} 0 \\ x'_i \end{pmatrix} P \qquad \text{for } i \in \{1, \ldots, l\}, \\ y_i &= P^T \begin{pmatrix} 0 \\ y'_i \end{pmatrix} P \qquad \text{for } i \in \{1, \ldots, l\}. \end{aligned}$$

3. If the diagonal of $A$ is all zero, but $A$ is has some non-zero entry, then swapping rows and columns with permutation matrix $P$ gives us

$$\begin{aligned} PAP^T = \begin{pmatrix} 0 & {v_1}^T & 1 \\ v_1 & A' & v_2 \\ 1 & {v_2}^T & 0 \end{pmatrix} &= \begin{pmatrix} 0 \\ v_1 \\ 1 \end{pmatrix} \begin{pmatrix} 1 & {v_2}^T & 0 \end{pmatrix} + \begin{pmatrix} 1 \\ v_2 \\ 0 \end{pmatrix} \begin{pmatrix} 0 & {v_1}^T & 1 \end{pmatrix} \\ &\quad + \begin{pmatrix} 0 & \vec{0}^T & 0 \\ \vec{0} & A^* & \vec{0} \\ 0 & \vec{0}^T & 0 \end{pmatrix} \end{aligned}$$

   Where $\vec{0}$ denotes the all zero vector, and $A^* = A' - v_1{v_2}^T - v_2{v_1}^T$, which is also symmetric. As in the previous cases, we recurse on our algorithm to

find the solution $u'_1, \ldots, u'_k, x'_1, \ldots, x'_{l-1}, y'_1, \ldots, y'_{l-1}$ for $A^*$. Our solution then becomes:

$$x_l = P^T \begin{pmatrix} 0 \\ v_1 \\ 1 \end{pmatrix} P, \qquad y_l = P^T \begin{pmatrix} 1 \\ v_2 \\ 0 \end{pmatrix} P,$$

$$u_i = P^T \begin{pmatrix} 0 \\ u'_i \\ 0 \end{pmatrix} P \qquad \text{for } i \in \{1, \ldots, k\}$$

$$x_i = P^T \begin{pmatrix} 0 \\ x'_i \\ 0 \end{pmatrix} P \qquad \text{for } i \in \{1, \ldots, l-1\},$$

$$y_i = P^T \begin{pmatrix} 0 \\ y'_i \\ 0 \end{pmatrix} P \qquad \text{for } i \in \{1, \ldots, l-1\}.$$

**Lemma 4.3.** *Sym_Rank_Base*$(A)$ *is correct and runs in time* $O(n^3)$.

*Proof.* We will prove correctness by induction over $n$. Since our recursion steps will decrease $n$ by 1 or 2, the base cases $n = 0$ and $n = 1$ suffice. If $n = 0$, $k = l = 0$, and we are done. If $n = 1$ and $\mathrm{rk}\,(A) = 1$, we return $u_1 = A$. If $n = 1$ and $\mathrm{rk}\,(A) = 0$ return return the empty set. In either way the algorithm finds a correct solution.

As the algorithm recurses on smaller matrices, we only need to show that the steps to create those smaller matrices are valid, after which the assumption that the algorithm is correct for all $(n-1) \times (n-1)$ matrices immediately tells us the entire algorithm is correct. In case 1, the algorithm terminates. In the other cases, as the smaller matrix can be created by taking a principal submatrix of the bigger matrix, and adding either a symmetric rank one matrix, or a matrix and its transpose, we find that this smaller matrix must also be symmetric. Thus, we can conclude that we can in fact recurse on these smaller matrices. Furthermore, as our solutions are built up from the solutions of the smaller matrices, one can verify with some simple computation that our new solutions are correct.

Let us now look at the relation between $k$, $l$, and $\mathrm{rk}\,(A)$. In case 1, $\mathrm{rk}\,(A) = 0$, and our output consists of zero vectors. Therefore, $k + 2l = \mathrm{rk}\,(A)$ holds.

In case 2, we claim that the rank is decreased by 1. This holds because we can find a basis for the row space of $PAP^T$ that contains $(1\ v^T)$, and where the remaining basis vectors all start with 0. Every row of $PAP^T$ can be written as a sum of these basis vectors. If the $j^{th}$ row of $A$ contains $(1\ v^T)$ in this sum, its first entry will be a 1, and

thus the $j^{th}$ entry of $(1\ v^T)$ is also 1. Thus, $PAP^T + (1\ v^T)^T(1\ v^T)$ removes $(1\ v^T)$ from the rows that contain it in their sum of basis vectors, meaning that all the rows in $PAP^T + (1\ v^T)^T(1\ v^T)$ can be written as sums of the remaining basis vectors. As we add a rank one matrix to $A$, we know that the rank cannot decrease by more than one. Therefore, we can conclude that $\mathrm{rk}\left(PAP^T + (1\ v^T)^T(1\ v^T)\right) = \mathrm{rk}\,(A) - 1$. So the rank is decreased by one, and $k$ is increased by 1. Under the assumption that the relation $(k-1) + 2l = \mathrm{rk}\,(A')$ holds, we find that $k + 2l = \mathrm{rk}\,(A)$ holds.

Finally, in case 3, we claim that the rank decreases by 2. This claim follows similar logic as the last claim, but now we consider a basis of $PAP^T$ that contains the vectors $(0\ {v_1}^T\ 1)$ and $(1\ {v_2}^T\ 0)$, and for which the remaining vectors start and end with a zero. Again we find that the remaining matrix can be written from the remaining basis vectors. Since we add two rank one matrices to $A$, the rank can decrease by at most two. This tells us that the rank decreases by two, and we find one more $x$-$y$ pair. Again, by the assumption that $k + 2(l-1) = \mathrm{rk}\,(A^*)$, we find that $k + 2l = \mathrm{rk}\,(A)$. Therefore, the relation must hold for any output of *Sym_Rank_Base*.

Lastly, let us consider the running time for each case. The first case is recognized in time $O(n^2)$, after which the algorithm terminates. We find the distinction between case 2 and 3 in time $O(n)$, after which either case takes up $O(n^2)$ time before recursing. Every recursion decreases $n$ by at least one, so we recurse no more than $n$ times. Therefore, the total algorithm runs in $O(n^3)$. ■

Of course, *Sym_Rank_Base* doesn't give us a symmetric rank decomposition, as the matrices ${x_i}{y_i}^T$ and ${y_i}{x_i}^T$ are not symmetric. So let us see how we can rewrite the sum over these pairs as a sum over symmetric rank one matrices:

**Lemma 4.4.** *Let $x_i, y_i$ for $i \in \{1, \ldots, l\}$ be $GF[2]$ column vectors of equal length. The following must hold:*

$$\begin{aligned}\sum_{i=1}^{l}\left({x_i}{y_i}^T + {y_i}{x_i}^T\right) = \sum_{i=1}^{l}\Bigg(&\left(\sum_{j=1}^{i-1}(x_j + y_j) + x_i\right)\left(\sum_{j=1}^{i-1}(x_j + y_j) + x_i\right)^T \\ &+ \left(\sum_{j=1}^{i-1}(x_j + y_j) + y_i\right)\left(\sum_{j=1}^{i-1}(x_j + y_j) + y_i\right)^T\Bigg) \\ &+ \left(\sum_{i=1}^{l} x_i + y_i\right)\left(\sum_{i=1}^{l} x_i + y_i\right)^T\end{aligned}$$

*Which shows that we can write the $lk$ rank one matrices on the left hand side as $2l + 1$ symmetric rank one matrices.*

*Proof.* We will prove this by induction. When $l = 1$, we see that

$$x_1 {y_1}^T + y_1 {x_1}^T = (x_1 + y_1)(x_1 + y_1)^T + x_1 {x_1}^T + y_1 {y_1}^T,$$

which completes our induction base. If the statement holds for $l - 1$ vector pairs, i.e. if

$$\begin{aligned}\sum_{i=1}^{l-1} \left(x_i {y_i}^T + y_i {x_i}^T\right) = \sum_{i=1}^{l-1} \Bigg( &\left(\sum_{j=1}^{i-1}(x_j + y_j) + x_i\right)\left(\sum_{j=1}^{i-1}(x_j + y_j) + x_i\right)^T \\ &+ \left(\sum_{j=1}^{i-1}(x_j + y_j) + y_i\right)\left(\sum_{j=1}^{i-1}(x_j + y_j) + y_i\right)^T\Bigg) \\ &+ \left(\sum_{i=1}^{l-1} x_i + y_i\right)\left(\sum_{i=1}^{l-1} x_i + y_i\right)^T\end{aligned}$$

Then we find that:

$$\begin{aligned}&\sum_{i=1}^{l} \Bigg( \left(\sum_{j=1}^{i-1}(x_j + y_j) + x_i\right)\left(\sum_{j=1}^{i-1}(x_j + y_j) + x_i\right)^T \\ &\quad + \left(\sum_{j=1}^{i-1}(x_j + y_j) + y_i\right)\left(\sum_{j=1}^{i-1}(x_j + y_j) + y_i\right)^T\Bigg) \\ &\qquad + \left(\sum_{i=1}^{l} x_i + y_i\right)\left(\sum_{i=1}^{l} x_i + y_i\right)^T = \sum_{i=1}^{l-1}\left(x_i {y_i}^T + y_i {x_i}^T\right) \\ &\qquad\qquad + \left(\sum_{j=1}^{l-1}(x_j + y_j) + x_l\right)\left(\sum_{j=1}^{l-1}(x_j + y_j) + x_l\right)^T \\ &\qquad\qquad + \left(\sum_{j=1}^{l-1}(x_j + y_j) + y_l\right)\left(\sum_{j=1}^{l-1}(x_j + y_j) + y_l\right)^T \\ &\qquad\qquad + \left(\sum_{i=1}^{l-1} x_i + y_i\right){x_l}^T + x_l\left(\sum_{i=1}^{l-1} x_i + y_i\right)^T \\ &\qquad\qquad + \left(\sum_{i=1}^{l-1} x_i + y_i\right){y_l}^T + y_l\left(\sum_{i=1}^{l-1} x_i + y_i\right)^T \\ &\qquad\qquad + x_l {y_l}^T + y_l {x_l}^T + x_l {x_l}^T + y_l {y_l}^T\end{aligned}$$

Under closer inspection, we find that

$$\left(\sum_{j=1}^{l-1}(x_j + y_j) + x_l\right)\left(\sum_{j=1}^{l-1}(x_j + y_j) + x_l\right)^T = \left(\sum_{i=1}^{l-1} x_i + y_i\right){x_l}^T + x_l\left(\sum_{i=1}^{l-1} x_i + y_i\right)^T + x_l {x_l}^T$$

$$\left(\sum_{j=1}^{l-1}(x_j+y_j)+y_l\right)\left(\sum_{j=1}^{l-1}(x_j+y_j)+y_l\right)^T=\left(\sum_{i=1}^{l-1}x_i+y_i\right){y_l}^T+y_l\left(\sum_{i=1}^{l-1}x_i+y_i\right)^T+y_l{y_l}^T$$

Combining these equations, we find that the lemma holds. ■

Lemma 4.4 gives us a way to turn the output of *Sym_Rank_Base* into a symmetric rank decomposition. This calls for a new algorithm:

*Sym_Rank_Advanced*$(A)$:
First we compute the solution to *Sym_Rank_Base*$(A)$, which we denote by $u_1,\ldots,u_k$, $x_1,\ldots,x_l$, $y_1,\ldots,y_l$. Then, if $l\neq 0$, we create $z_1,\ldots,z_{2l+1}$ out of $x_1,\ldots,x_l$, $y_1,\ldots,y_l$, as described in Lemma 4.4. And we find that

$$\sum_{i=1}^{k}u_i{u_i}^T+\sum_{j=1}^{2l+1}z_j{z_j}^T=A$$

Thus $u_1,\ldots,u_k,z_1,\ldots,z_{2l+1}$ gives us a symmetric rank decomposition of $A$. As $k+2l=\mathrm{rk}\,(A)$, we find that the size of our symmetric rank decomposition is at most $\mathrm{rk}\,(A)+1$, with strict inequality if and only if $l=0$.

The correctness of this algorithm follows from the correctness of *Sym_Rank_Base*, and the correctness of Lemma 4.4. Note that creating the vectors $z_1,\ldots,z_{2l+1}$ can be done in time $O(n^2)$. Therefore, we can conclude that the running time is identical to *Sym_Rank_Base*, meaning it is also $O(n^3)$. The existence of this algorithm implies the following result:

**Theorem 4.5.** *Let $A$ be a symmetric $GF[2]$ matrix. It holds that*

$$\mathrm{rk}\,(A)\leq\mathrm{rk}_{\mathbb{S}}\,(A)\leq\mathrm{rk}\,(A)+1$$

*Where $A_G$ denotes the adjacency matrix of $G$.*

*Proof.* The lower bound in this theorem is a restating of Lemma 4.2. The correctness of *Sym_Rank_Advanced* tells us that we can always find a symmetric rank decomposition for $A$ of size at most $\mathrm{rk}\,(A)+1$, which gives us the upper bound. ■

So, we have found an approximation algorithm that is very efficient, as no solution it generates can be larger than the optimal value plus one. However, we can still do better, which we will prove in the next section.

### 4.1.2 Zero-Diagonal Matrices

In our approximation algorithm, we saw that the sub-optimality of our approximation came from handling matrices where all diagonal entries are equal to 0. We call such matrices *zero-diagonal matrices*. In this section, we will focus on this type of matrices. First we will prove that given a symmetric zero-diagonal matrix $A$, there exists no symmetric rank one matrix $B$ such that $\mathrm{rk}\,(A+B) < \mathrm{rk}\,(A)$, which is done in Lemma 4.7. After this, we show in Lemma 4.8 that the $\mathrm{rk}\,(A)$ for any symmetric zero-diagonal matrix $A$ must be even. We use this insight to show that we can deal with symmetric zero-diagonal matrices in a more efficient way. Finally, we create a new algorithm, whose correctness we prove in the proof of Theorem 4.1. But first, let us make the following observation about symmetric zero-diagonal $GF[2]$ matrices:

Let $A$ be a symmetric zero-diagonal $GF[2]$ matrix. Since $A$ is zero-diagonal, we know that if $A[i,j] = 1$, then $i \neq j$. Furthermore, the symmetry of $A$ tells us that if $A[i,j] = 1$, then also $A[j,i] = 1$. This means that every entry of $A$ that is equal to 1 has a unique pair. Therefore, there cannot be an odd number of 1's in $A$. For future reference, we express this property in the following observation:

**Observation 3.** *Let $A$ be a symmetric zero-diagonal $GF[2]$ matrix. The total number of 1's in $A$ is even.*

Now consider the following result on principal sub-matrices:

**Lemma 4.6.** *Let $A$ be a symmetric $GF[2]$ $n \times n$ matrix such that $\mathrm{rk}\,(A) = r < n$. Then there exists an $I \subset \{1,\ldots,n\}$, such that $\mathrm{rk}\,(A[I,I]) = r$, where $A[I,I]$ denotes the principal submatrix on $I$.*

*Proof.* Since the size of $I$ is not specified, we can assume that $|I| = n-1$, as any smaller $I$ would trivially extend to an $I$ of size $n-1$. Since $r < n$, there exist a non-empty set of linearly dependent rows. Since we are working in $GF[2]$, this tells us there exists a non-empty subset of rows $\{\alpha_1,\ldots,\alpha_l\}$ such that

$$\sum_{i=2}^{l} \alpha_i = \alpha_1. \tag{4.1}$$

Let $j$ be the index corresponding to $\alpha_1$, and pick $I = \{1,\ldots,j-1,j+1,\ldots,n\}$. If $\mathrm{rk}\,(A[I,I]) = \mathrm{rk}\,(A)$, then $I$ is of the desired form.

Else, we know that $\mathrm{rk}\,(A[I,I]) < \mathrm{rk}\,(A)$. That means there exists a non-empty subset of columns $\{\beta_1, \ldots, \beta_k\}$ that are linearly independent in $A$, but sum up to 0 when restricted to $A[I,I]$. This implies that

$$\sum_{i=1}^{k} \beta_i = e_j,$$

where $e_j$ denotes the $j^{th}$ unit vector. However, this contradicts equation (4.1): We know $\sum_{i=1}^{k} \beta_i$ should have at least one other non-zero element within the rows $\{\alpha_2, \ldots, \alpha_l\}$, because otherwise the sum over $\{\alpha_2, \ldots, \alpha_l\}$ is not equal to $\alpha_1$ in one of the $\beta_i$ columns. Therefore, we can conclude that $\mathrm{rk}\,(A[I,I]) = \mathrm{rk}\,(A)$, proving our lemma. ■

This result is necessary to prove a more interesting result on zero-diagonal matrices.

**Lemma 4.7.** *Given symmetric $GF[2]$ matrix $A$, with $\mathrm{rk}\,(A) = r > 0$, such that all diagonal entries of $A$ are 0's. There exists no symmetric $GF[2]$ rank one matrix $B$ for which $\mathrm{rk}\,(A+B) = r-1$.*

*Proof.* We will prove this by induction over the size of $A$. In the base case

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

it is clear that $\mathrm{rk}\,(A) = 2$. There are three possibilities for $B$:

$$B_1 = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \qquad B_2 = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \qquad B_3 = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$
$$A + B_1 = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \qquad A + B_2 = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \qquad A + B_3 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

The resulting matrices all still have rank 2, so the lemma holds in the base case.

Now suppose that the lemma holds for all matrices smaller than $n \times n$, and let $A$ be a symmetric $GF[2]$ $n \times n$ matrix. If $\mathrm{rk}\,(A) < n$, then by Lemma 4.6, we can find $I \subset \{1, \ldots, n\}$ such that $\mathrm{rk}\,(A[I,I]) = \mathrm{rk}\,(A)$. By our induction hypothesis, there exists no symmetric rank one matrix $B$ such that $\mathrm{rk}\,(A[I,I] + B) = \mathrm{rk}\,(A[I,I]) - 1$, and as $A[I,I]$ is a submatrix of $A$ of the same rank, we know that there cannot exist a symmetric rank one matrix $B'$ such that $\mathrm{rk}\,(A + B') = \mathrm{rk}\,(A) - 1$.

This leaves the case where $A$ is of full rank. Suppose, for the sake of contradiction, that there exists an $n \times n$ symmetric zero-diagonal $GF[2]$ rank $n$ matrix $A$ for which

there exists a matrix symmetric rank one $B$ such that $\mathrm{rk}\,(A+B) = n-1$. Without loss of generality, $B = uu^T$ where

$$u^T = (\underbrace{1,\ldots,1}_{m \text{ times}},0,\ldots,0).$$

Let $I \subseteq \{1,\ldots,n\}$ be a non-empty set of rows that sum up to 0 in $A+B$, and define $I^- = I \cap \{1,\ldots,m\}$ and $I^+ = I \setminus \{1,\ldots,m\}$.

Claim: $|I^-| \neq 0$, and $|I^+| \neq 0$.

Proof of Claim: If $|I^-| = 0$, then $A$ cannot be of full rank; The rows $I^+$ are identical in $A$ and $A+B$, and as they sum up to 0 in $A+B$, they must be linearly dependent in $A$, contradicting the argument that $A$ is a full rank matrix. So $|I^-| \neq 0$.

Suppose for the sake of contradiction that $|I^+| = 0$. This means that the rows $I^-$ sum up to 0 in $A+B$. This must mean that the number of 1's in every column of $(A+B)[I^-,I^-]$ is even. As addition of $B[I^-,I^-]$ is an all 1 matrix, we know that

$$(A+B)[i,j] \equiv A[i,j]+1 \pmod 2 \qquad \forall i,j \in I^-.$$

Therefore, the number of 0's in every column of the submatrix of $A[I^-,I^-]$ is even. Since the rows $I^-$ in $A$ are linearly independent before addition of $B$, we know that there exists at least one column with an odd amount of 1's in the rows $I^-$. We know that this column must itself lie in $I^-$, as otherwise this column wouldn't be affected by the addition of $B$. Thus $A[I^-,I^-]$ has a column with an odd amount of 1's, and an even amount of 0's. This implies that the amount of entries in this column are odd, and therefore that $|I^-|$ is odd.

Consider again the principal submatrix of $A[I^-,I^-]$. We know the amount of 0's in every column is even. We also know that this submatrix is symmetric and has diagonal completely zero. By Observation 3, we thus know that the amount of 1's in $A[I^-,I^-]$ must be even. But, as $|I^-|$ is odd, there is an odd amount of total entries in this principal submatrix, contradicting the statement that the number of 0's, and the number of 1's are both even. Therefore, $|I^+| \neq 0$. □

Consider the principal submatrices of $A$ and $A+B$ on $I$, denoted by $A_I$ and $A'_I$, respectively. These submatrices must be symmetric, and $A_I$ must have diagonal completely zero. As $|I^-| \neq 0$ and $|I^+| \neq 0$, we can write $A_I$ in the following way:

$$A_I = \begin{pmatrix} A^- & (A^*)^T \\ A^* & A^+ \end{pmatrix}, \qquad A'_I = \begin{pmatrix} \overline{A^-} & (A^*)^T \\ A^* & A^+ \end{pmatrix}.$$

Where $A^-$ and $A^+$ denote the principal submatrices on $I^-$ and $I^+$, respectively, $A^*$ denotes the remaining matrix, and $\overline{A^-}$ is the matrix defined by

$$\overline{A^-}[i,j] \equiv A^-[i,j] + 1 \pmod 2$$

The approach of our proof is as follows; We will show that the number of ones in $A^*$ has to be odd by observations on $A^-$. After that, we will show that this leads to a contradiction in the parity of the number of ones in $A^+$.

Firstly, we want to show that $|I^-|$ must be odd. We know that for every column of $A'_I$, the number of 1's should be even. This implies that for every column, the number of 0's in $A^-$ (which is equal to the number of 1's in $\overline{A^-}$ in this column) plus the number of 1's in $A^*$ should be even. As the last $|I^+|$ columns of $A_I$ and $A'_I$ are identical, and the rows of $A_I$ do not sum up to 0, we know that one of the first $|I^-|$ columns of $A_I$ contains an odd number of 1's. This implies that for this column, the number of 0's in $A^-$ plus the number of 1's in $A^*$ is even, whilst the number of 1's in $A^-$ plus the number of 1's in $A^*$ is odd. Thus, in this column, the number of 1's and the number of 0's in $A^-$ have different parity. As $|I^-|$ is equal to the total amount of entries in this column of $A^-$, which is equal to the number of 1's in this column plus the number of 0's in this column, we know that $|I^-|$ must be odd.

Secondly, we will show that the number of 1's in $A^*$ must be odd. Since $A^-$ is symmetric and zero-diagonal, Observation 3 tells us that there is an even number of 1's in $A^-$. As $|I^-|$ is odd, the total number of entries in $A^-$ is odd, implying that there is an odd number of 0's. Since we know that in all the first $|I^-|$ columns the amount of 0's in $A^-$ plus the amount of 1's in $A^*$ should be even, we know that the total number of 0's in $A^-$ plus the total number of 1's in $A^*$ should also be even, as the sum of several even numbers must itself be even. As we know that the number of 0's in $A^-$ is odd, the number of 1's in $A^*$ must also be odd.

Finally, we show what this implies for the amount of 1's in $A^+$, and how that leads to a contradiction. As the rows of $A'_I$ should sum up to 0, we know that for every column, the amount of 1's in $(A^*)^T$ plus the amount of 1's in $A^+$ should be even. As the sum of multiple even numbers must also be even, we know that the number of 1's in $(A^*)^T$ plus the number of 1's in $A^+$ is even. We know that the total number of 1's in $A^*$ is odd, which tells us that the total number of 1's in $A^+$ must also be odd. However, this cannot be the case, as $A^+$ is symmetric and zero-diagonal, which, by observation 3, implies that the total number of 1's in $A^+$ must be even.

Therefore we can conclude that for any full rank symmetric zero-diagonal $GF[2]$ matrix $A$, there cannot exist a symmetric rank one matrix $B$ such that $\mathrm{rk}\,(A+B) = \mathrm{rk}\,(A) - 1$. As this is the last case for our induction, the lemma holds. ■

**Corollary 4.7.1.** *Let $A$ be a symmetric $GF[2]$ with $\mathrm{rk}(A) = r > 0$, such that all diagonal entries of $A$ are 0's. Then it holds that $\mathrm{rk}_{\mathbb{S}}(A) = r + 1$*

*Proof.* By Lemma 4.7, we know that there is no symmetric $GF[2]$ rank one matrix $B$ such that $\mathrm{rk}(A + B) = r - 1$. This tells us that in any symmetric rank decomposition, the 'first' matrix can't decrease the rank. After this 'first' matrix, we thus need at least $r$ more matrices to get the rank to zero. This tells us that $\mathrm{rk}_{\mathbb{S}}(A) \geq r + 1$. Using *Sym_Rank_Advanced*$(A)$, we can find a symmetric rank decomposition for $A$ of size $r + 1$, implying that $\mathrm{rk}_{\mathbb{S}}(A) \leq r + 1$. Combining these results tells us that $\mathrm{rk}_{\mathbb{S}}(A) = r + 1$. ■

We know now that a zero-diagonal matrix will always have symmetric rank one larger than its rank. However, this does not imply that finding a zero-diagonal matrix whilst performing *Sym_Rank_Advanced* also guarantees that the symmetric rank of the input matrix is one larger than its regular rank. To show this, we first need to determine the parity of the rank of a zero-diagonal matrix, which turns out to always be even:

**Lemma 4.8.** *Given symmetric $GF[2]$ zero-diagonal matrix $A$. Then $\mathrm{rk}(A)$ is even.*

*Proof.* We will prove this by induction over the size of $A$. In the base case, $A$ is a one by one matrix. If $A$ is a one by one matrix, then clearly the only option for $A$ is $(0)$, which clearly has rank $0$. As zero is even, the lemma holds in the base case.

Suppose the lemma holds for all $(n-1) \times (n-1)$ matrices, and consider an $n \times n$ symmetric $GF[2]$ matrix $A$. We can write $A$ as

$$A = \begin{pmatrix} A' & u \\ u^T & 0 \end{pmatrix}$$

for some $(n-1) \times (n-1)$ symmetric $GF[2]$ matrix $A'$, and some $(n-1)$-length $GF[2]$ row vector $u$. By our induction hypothesis, $\mathrm{rk}(A')$ is even. Suppose for the sake of a contradiction that $\mathrm{rk}(A)$ is odd. This implies one of the following statements:

1. $(u^T, 0)$ can be written as the sum of rows of $(A', u)$, and $\mathrm{rk}(A', u) = \mathrm{rk}(A') + 1$.

2. $(u^T, 0)$ cannot be written as the sum of rows of $(A', u)$, and $\mathrm{rk}(A', u) = \mathrm{rk}(A')$.

Observe that these statements cover all options where $\mathrm{rk}(A)$ is odd. The missing statements are "$(u, 0)$ cannot be written as the sum of rows of $(A', u)$, and $\mathrm{rk}(A', u) = \mathrm{rk}(A') + 1$", which would imply that $\mathrm{rk}(A) = \mathrm{rk}(A') + 2$, and "$(u, 0)$ can be written

as the sum of rows of $(A', u)$, and $\mathrm{rk}\,(A', u) = \mathrm{rk}\,(A')$", which would imply that $\mathrm{rk}\,(A) = \mathrm{rk}\,(A')$. Both these options would result in $A$ having even rank.

Let us consider the first statement. Since $(u^T, 0)$ can be written as the sum of rows of $(A', u)$, we know that $u^T$ can be written as the sum of rows of $A'$. As $A'$ is symmetric, this also means that $u$ can be written as a sum of the columns of $A'$. But then we know that $\mathrm{rk}\,(A', u) = \mathrm{rk}\,(A')$, contradicting the statement. Therefore this statement cannot be true.

Now consider the second statement. Following similar logic, the fact that $\mathrm{rk}\,(A', u) = \mathrm{rk}\,(A')$ tells us that $u$ can be written as a sum of columns of $A'$, and therefore by symmetry that $u^T$ can be written as a sum of rows of $A'$. Let $I$ be a set of row indices such that the rows of $A'[I, \cdot]$ (the rows of $A'$ with indices in $I$) sum up to $u^T$. Since the rows of $(A', u)[I, \cdot]$ do not sum up to $(u^T, 0)$, they must sum up to $(u^T, 1)$. This tells us that there is an odd amount of 1's in $u[I]$ (vector $u$ restricted to indices in $I$).

We know that $u$ is equal to the sum of the columns of $A'[\cdot, I]$. Therefore, $u[I]$ is equal to the sum of the columns of $A'[I, I]$. If an element of $u[I]$ is 1, that means that the corresponding row of $A'[I, I]$ contains an odd number of 1's. The fact that $u[I]$ has an odd number of 1's thus tells us that the number of rows of $A'[I, I]$ that contain an odd number of 1's, must itself be odd. As every other element of $u[I]$ must be 0, every other row of $A'[I, I]$ must have an even number of 1's.

Consider now the total number of 1's in $A'[I, I]$. We know this must be equal to a sum of an odd number of odd numbers (corresponding to the rows where $u[I]$ is 1), plus a sum over a set of even numbers (corresponding to the rows where $u[I]$ is 0). This tells us that the total number of 1's in $A'[I, I]$ must be odd. But, as $A'[I, I]$ is symmetric and zero-diagonal, this contradicts Observation 3. Therefore, this second statement also cannot be true. As we've shown that these two statements cover all scenarios in which $\mathrm{rk}\,(A)$ is odd, we know that our induction step is correct, and therefore that the lemma holds. ■

The following lemma should allow us to simplify some problem instances:

**Lemma 4.9.** *Let $A$ be a symmetric $GF[2]$ matrix. Let $a_1, \ldots, a_n$ denote the columns of $A$. Suppose there exist $a_i = a_j$ for $i \neq j$. Let $I = \{1, \ldots, n\} \setminus i$. Any symmetric rank decomposition of $A[I, I]$ translates to a symmetric rank decomposition of $A$, and $\mathrm{rk}_{\mathbb{S}}\,(A[I, I]) = \mathrm{rk}_{\mathbb{S}}\,(A)$.*

*Proof.* Without loss of generality, we assume that $i = n$. Let $u_1, \ldots, u_k$ be a symmetric rank decomposition for $A[I, I]$. Define $u'_1, \ldots, u'_k$ as

$$u'_l = (u_l[1], \ldots, u_l[n-1], u_l[j])^T \qquad \text{for } l \in \{1, \ldots, k\}.$$

And consider $B = \sum_{l=1}^{k} (u'_l)(u'_l)^T$, and let $b_1, \ldots, b_n$ denote the columns of $B$. Note that by construction, $B[I, I] = A[I, I]$. Also, note that $B$ is symmetric. Consider the $i^{th}$ column of $B$, denoted by $b_i$, and observe that it can be written as

$$b_i = \sum_{l=1}^{k} u'_l[i] \cdot (u'_l) = \sum_{l=1}^{k} u'_l[j] \cdot (u'_l) = b_j$$

Note that $b_j[I] = a_j[I]$, and $b_j[i] = b_j[j] = a_j[j] = a_j[i]$, meaning that $b_i = b_j = a_j = a_i$. So, $B[I, I] = A[I, I]$, the $i^{th}$ column of $B$ and $A$ are equal, and by symmetry the $i^{th}$ row of $B$ and $A$ are also equal. Thus $B = A$. Therefore, we have found a symmetric rank decomposition of $A$. Note that we don't increase the number of rank one matrices in our decomposition, and thus that there exists a symmetric rank decomposition of $A$ of size $\mathrm{rk}_{\mathbb{S}}(A[I, I])$. Thus, $\mathrm{rk}_{\mathbb{S}}(A) \leq \mathrm{rk}_{\mathbb{S}}(A[I, I])$. As $A[I, I]$ is a principal submatrix of $A$, we know that $\mathrm{rk}_{\mathbb{S}}(A[I, I]) \leq \mathrm{rk}_{\mathbb{S}}(A)$. Combining these statements, we can conclude that $\mathrm{rk}_{\mathbb{S}}(A) = \mathrm{rk}_{\mathbb{S}}(A[I, I])$. ■

Now we can finally show how to efficiently deal with zero-diagonal matrices encountered whilst performing *Sym_Rank_Advanced*:

**Lemma 4.10.** *Let $A$ be a symmetric $GF[2]$ matrix with at least one 1 on its diagonal. Let $a_1, \ldots, a_n$ denote the $n$ columns of $A$. Let $I$ bet the indices of the rows of $A$ that have a 1 on their diagonal position. If $A + a_i a_i^T$ is a zero-diagonal matrix for all $i \in I$, then $\mathrm{rk}_{\mathbb{S}}(A) = \mathrm{rk}(A)$.*

*Proof.* Since there is no $i \in I$ such that $A + a_i a_i^T$ contains a 1 on the diagonal, we claim the following:

Claim: All $a_i$ with $i \in I$ are indicator vectors of $I$, i.e. $a_i[j] = 1$ if and only if $j \in I$.

Proof of Claim: Suppose that there are $i, j \in I$ such that $a_i[j] = 0$. We then know that $j \neq i$ by definition of $I$. By symmetry, this then tells us that $a_j[i] = 0$. Consider $A + a_j a_j^T$. As $a_j[i] = 0$, we know that $(a_j a_j^T)[i, i] = 0$. But then $(A + a_j a_j^T)[i, i] = A[i, i] = 1$, which contradicts the statement that $A + a_i a_i^T$ is a zero-diagonal matrix for all $i \in I$.

Now suppose that there is an $i \in I$ and a $j \notin I$ such that $a_i[j] = 1$. Then we know that ${a_i a_i}^T[j,j] = 1$. Therefore, $(A + a_i {a_i}^T)[j,j] = 0 + 1 = 1$. This again contradicts the statement that $A + a_i {a_i}^T$ is a zero-diagonal matrix for all $i \in I$. □

So, all the $a_i$ for $i \in I$ are duplicates. By Lemma 4.9, we know that we can remove column duplicates whilst preserving the size of the optimal solution. Let us denote the reduced matrix by $A^*$, and observe that (after reordering the indices) this matrix must have the following form:

$$A^* = \begin{pmatrix} A' & \vec{0} \\ \vec{0}^T & 1 \end{pmatrix}$$

Where $A'$ is some symmetric zero-diagonal $GF[2]$ matrix and $\vec{0}$ is the all zero column vector. Let us denote $\mathrm{rk}\,(A') = r$. By Lemma 4.8, we know that the $r$ must be even. We know that *Sym_Rank_Advanced*$(A')$ will give us column vectors $\{u_1, \ldots, u_{r+1}\}$ such that

$$A' = \sum_{i=1}^{r+1} u_i {u_i}^T.$$

If we define vectors $\{v_1, \ldots, v_{r+1}\}$ as

$$v_i = \begin{pmatrix} u_i \\ 1 \end{pmatrix} \qquad \text{for } i \in \{1, \ldots, r+1\},$$

Claim: $\sum_{i=1}^{r+1} v_i {v_i}^T = A^*$.

Proof of Claim: To see that this holds, first note that:

$$\sum_{i=1}^{r+1} \begin{pmatrix} u_i \\ 0 \end{pmatrix} \begin{pmatrix} {u_i}^T & 0 \end{pmatrix} = \begin{pmatrix} A' & \vec{0} \\ \vec{0}^T & 0 \end{pmatrix}$$

Next, observe that changing the last positions of the vectors will only affect the last row and the last column of the resulting matrix. Also, the bottom right position will be a 1, as $r+1$ is odd, and all vectors $v_i$ end with a 1. This tells us that

$$\sum_{i=1}^{r+1} v_i {v_i}^T = \begin{pmatrix} A' & x \\ x^T & 1 \end{pmatrix}.$$

Suppose for the sake of contradiction that $x$ is not the all zero vector. Then, there exists a $j$ such that $x[j] = 1$. This tells us that the amount of vectors $v_i$ that contain a 1 in the last position, and in the $j^{th}$ position must be odd. As all vectors $v_i$ end on a 1, this tells us that the amount of vectors $u_i$ that have a 1 in the $j^{th}$ position must

be odd. But, this gives us a contradiction, as this would imply that $A'$ has a 1 on the $j^{th}$ diagonal position. Therefore, our claim holds. □

As $\mathrm{rk}(A^*) = r + 1$, we can conclude that the symmetric rank decomposition $\{v_1, \ldots, v_r\}$ is minimal, and thus that $\mathrm{rk}_{\mathbb{S}}(A) = \mathrm{rk}_{\mathbb{S}}(A^*) = \mathrm{rk}(A^*) = \mathrm{rk}(A)$. ■

Now we have everything in place to prove Theorem 4.1. For clarity, let us restate the theorem:

**Theorem 4.1.** *Let $A$ be a symmetric $GF[2]$ matrix, such that $A$ is not the all zero matrix. If $A$ has no 1 on its diagonal, then $\mathrm{rk}_{\mathbb{S}}(A) = \mathrm{rk}(A) + 1$. Otherwise, $\mathrm{rk}_{\mathbb{S}}(A) = \mathrm{rk}(A)$.*

*Proof.* We will prove this by induction over $\mathrm{rk}(A)$. If $\mathrm{rk}(A) = 0$ then $A$ is the all-zero matrix, which is not covered by our theorem as it is a trivial case. Instead, let our base case be $\mathrm{rk}(A) = 1$. We know that $A$ is a symmetric rank one matrix, and therefore that $A$ is a symmetric rank decomposition for itself. By Lemma 4.2, we know $\mathrm{rk}_{\mathbb{S}}(A) \geq 1$. And thus, we can conclude that $\mathrm{rk}_{\mathbb{S}}(A) = 1$, and our theorem holds in the base case.

Suppose that our induction hypothesis holds for all $A'$ with $\mathrm{rk}(A') < r$. Let $A'$ be a symmetric $GF[2]$ matrix such that $\mathrm{rk}(A) = r$. If $A$ has an all 0 diagonal, Corollary 4.7.1 tells us that $\mathrm{rk}_{\mathbb{S}}(A) = r + 1$.

Instead, suppose that $A$ has at least one 1 on its diagonal. Denote the columns of $A$ that have a non-zero diagonal entry by $c_1, \ldots, c_k$. By the correctness of *Sym_Rank_Base*$(A)$, proved in Lemma 4.3, we know that $\mathrm{rk}\left(A + c_i {c_i}^T\right) = r - 1$ for all $i \in \{1, \ldots, k\}$.

If there exists an $i \in \{1, \ldots, k\}$ such that $A + c_i {c_i}^T$ has a non-zero diagonal entry, then we can use our induction hypothesis to conclude that $\mathrm{rk}_{\mathbb{S}}\left(A + c_i {c_i}^T\right) = r - 1$. Since $A + c_i {c_i}^T$ is one symmetric rank one matrix away from $A$, we know that $\mathrm{rk}_{\mathbb{S}}(A) \leq r$. Applying Lemma 4.2, we find $\mathrm{rk}_{\mathbb{S}}(A) \geq r$, meaning that we can conclude that $\mathrm{rk}_{\mathbb{S}}(A) = r$.

If no such $i$ exists, we know that we are in the case described in Lemma 4.10. Thus, we can apply this lemma, and find that $\mathrm{rk}_{\mathbb{S}}(A) = r$, completing our induction step. This completes the proof. ■

Theorem 4.1 tells us that if we are only interested in the symmetric rank, all we have to do is compute the rank of our matrix, and check if the diagonal is completely

0. By Guassian elimination, this can be done in time $O(n^3)$. However, we know that this is not the most efficient way to determine matrix rank. For example, we can compute the rank of a matrix over any field in time $O(n^\omega)$, where $\omega$ is the matrix multiplication exponent [3, 17]. We know that $\omega < 2.3728639$ [21], showing that this algorithm is significantly more efficient than Gaussian elimination. We also know that there exists a probabilistic algorithm to determine $\min\{\mathrm{rk}\,(A)\,, k\}$ in an expected number of $O(n^2k^{\omega-2})$ steps [27], meaning we can solve $\widehat{PC_{\mathcal{G}}}(G,k)$ in time $O(n^2k^{\omega-2})$. Note that this is far from a complete study on which algorithm for determining matrix rank is the most efficient in our setting. For an overview of various algorithms to determine matrix rank, we recommend Cheung et al. [5].

However, in some cases we might be interested in finding the symmetric rank decomposition. To achieve this we present the following algorithm, based on the proof of Theorem 4.1. As the algorithm follows the proof almost exactly, we will not provide a separate proof of correctness. In this pseudocode, we assume $A$ to be an $n \times n$ symmetric $GF[2]$ matrix of rank $r$. Note that if a return statement is called, we will no longer perform any other steps within that iteration of the algorithm.

**Algorithm *Sym_Rank_Complete*($A$)**

1. $I := \{i : A[i,i] = 1\}$
2. **if** $I = \emptyset$
3. **then return** *Sym_Rank_Advanced*($A$)
4. **for** $i \in I$
5. %We denote the columns of $A$ by $a_1, \ldots, a_n$
6. $B = A + a_i {a_i}^T$
7. **if** $\exists j$ such that $B[j,j] = 1$
8. **then return** $\{a_i,$ *Sym_Rank_Complete*($B$)$\}$
9. %If there is no such $i \in I$:
10. $J = \{1, \ldots, n\} \setminus I$
11. $\{v_1, \ldots, v_{r+1}\} =$ *Sym_Rank_Advanced*($A[J,J]$)
12. **return** $\{u_i \mid i \in \{1, \ldots, r+1\}, u_i[j] = v_i[j']$ if $J[j'] = j$, and $u_i[j] = 1$ if $j \notin J\}$

Let us now take a look at the running time of *Sym_Rank_Complete*($A$). We start by checking which diagonal entries are 1, which is done in time $O(n)$. If there are no such entries, we call for *Sym_Rank_Advanced*($A$), which runs in time $O(n^3)$. Else, we enter a for loop, with at most $n$ repetitions. Computing $B$ will take $O(n^2)$ time, and checking the diagonal will again take $O(n)$ time. If this $B$ satisfies the conditions, this iteration of the algorithm stops. If no such $B$ is found, we again call *Sym_Rank_Advanced*, running in time $O(n^3)$, and alter the solution we achieve from this. Since the for loop has atmost $n$ repetitions, requires at $O(n^2)$ operations, and the whole algorithm recurses a total of $r$ times, our running time becomes $O(rn^3)$.

To conclude this section, note that we now have an efficient algorithm to compute the symmetric rank of any symmetric $GF[2]$ $n \times n$ matrix $A$ with $\mathrm{rk}(A) > 0$, as well as an algorithm to find an optimal symmetric rank decomposition. By Corollary 4.1.1, this also means that we have an efficient algorithm for $|PC_{\mathcal{G}}(G)|$ and $PC_{\mathcal{G}}(G)$.

### 4.1.3 Further Results on Symmetric Rank

In this subsection, we give some results on symmetric rank that did not lead to a result on our partial complement problem, but are interesting nonetheless. Namely, we want to take a closer look at symmetric rank decompositions:

**Lemma 4.11.** *Let $A$ be a symmetric $GF[2]$ matrix, with at least one non-zero diagonal entry. Denote $\mathrm{rk}(A) = r$, and let $u_1, \ldots, u_r$ be a symmetric rank decomposition of $A$. Then $u_1, \ldots, u_r$ are linearly independent.*

*Proof.* Note that we can write the $j^{th}$ row of $A$, which we denote by $A[j, \cdot]$, as

$$A[j, \cdot] = \sum_{i=1}^{r} u_i[j] \cdot {u_i}^T$$

Where $u_i[j]$ denotes the $j^{th}$ element of $u_i$. Therefore, we know that every row of $A$ is a linear combination of the $u_i$'s, i.e. that $u_1, \ldots, u_r$ forms a basis for the row space of $A$. Since $\mathrm{rk}(A) = r$, we know that this basis is of minimal size, implying that $u_1, \ldots, u_r$ are linearly independent. ■

This result lead us to the following Theorem:

**Theorem 4.12.** *Let $A$ be a symmetric $GF[2]$ matrix, with at least one non-zero diagonal entry. Denote $\mathrm{rk}(A) = r$, and let $u_1, \ldots, u_r$ be a symmetric rank decomposition of $A$. Given some $GF[2]$ vector $v$, it holds that $\mathrm{rk}\left(A + vv^T\right) = r - 1$, then there exists an $I \subseteq \{1, \ldots, r\}$ such that*

$$v = \sum_{i \in I} u_i.$$

*Proof.* Note that by Lemma 4.11, the $u_i$'s should be linearly independent. As there are $r = \mathrm{rk}(A)$ such vectors, they must thus form a basis for the row space of $A$. We claim that any $v$ with the property that $\mathrm{rk}\left(A + vv^T\right) = r - 1$ must be part of the row space of $A$. Suppose for the sake of contradiction that we have a $v$ satisfying this property that is not part of the row space of $A$. Then we know that

$$(A + vv^T)[j, \cdot] = A[j, \cdot] + v[j]v^T.$$

But then we cannot possibly decrease the rank, as we add a row that is not yet part of the rowspace. Thus $v$ must indeed be part of the row space. This implies $v$ must be a linear combination of the $u_i$'s, i.e. there exists an $I \subseteq \{1, \ldots, r\}$ such that $v = \sum_{i \in I} u_i$. ■

When developing this theorem, we had reason to believe that the result was actually stronger than what we have been able to prove here. Namely, by conducting experiments, we thought that the $I$ mentioned in the theorem needed to be of odd cardinality, and that any such $I$ would also give us a $v$ that decreases the rank. However, though all of the examples we tried fitted this stronger theorem, we could find no argumentation to support this claim. In Chapter 7, one of our open questions considers this theorem, and whether or not the stronger result is provable.

## 4.2 $P_3^*$-Free Reflexive Unigraphs of at most $d$ Connected Components

In this section, we are interested in a the graph family $\widehat{\mathcal{H}}^d$, which we define below. Let $P_3^*$ be the reflexive unigraph variant of $P_3$, i.e. a path on 3 vertices, where every vertex has a loop. Note that the family of $P_3^*$-free reflexive unigraphs, which we denote by $\widehat{\mathcal{H}}$, is a loop extension family of $P_3$-free simple graphs, denoted by $\mathcal{H}$.

**Definition.** Let $\widehat{\mathcal{H}}^d$ be defined as the family of $P_3^*$-free reflexive unigraphs of at most $d$ connected components. In other words, $\widehat{\mathcal{H}}^d$ consists of the graphs of $\widehat{\mathcal{H}}$ consisting of at most $d$ connected components.

Note that $\widehat{\mathcal{H}}^d$ is a loop extension family of $\mathcal{H}^d$, the family of $P_3$-free simple graphs with at most $d$ components. The strategy for $\widehat{\mathcal{H}}^d$ is as follows: Suppose that we have an oracle that, given an instance of $\widehat{PC_{\widehat{\mathcal{H}}^d}}(G)$, could tell us what the connected components of the solution graph are. We could then take $G$, and take partial complements on the at most $d$ connected components, and denote our new graph by $G'$. After this, we could search for the optimal solution to turn $G'$ into an edgeless graph, and this solution would then be the optimal solution for $\widehat{PC_{\widehat{\mathcal{H}}^d}}(G)$. However, we do not have such an oracle, so we will have to find the optimal $d$ components. In this section, we will discuss a method to find these components.

If we want to find the optimal way to divide the vertices of our graph into $d$ sets, we know that we will find it if we consider all the possible ways our vertices could be divided over $d$ sets. However, without restricting the amount of vertices, this will be equal to $d^n/d!$ ways, which is to large for an FPT algorithm. Therefore, we

must first say something about the distinct vertices. This requires the concept of *twin vertices*.

**Definition.** Let $G = (V, E)$ be a graph. Two vertices $u, v \in V$ are called *twins*, or *twin vertices*, if $N_G(u) = N_G(v)$.

Note that if a vertex $u$ has a loop, then $u \in N_G(u)$. Therefore, twin vertices can be adjacent in unigraphs, whereas this is not possible in simple graphs.

**Lemma 4.13.** *Let $G$ be an instance of $\widehat{PC_{\widehat{\mathcal{H}}^d}}(G)$, and let $u, v \in V(G)$ be twin vertices. Then there exists an optimal solution of $\widehat{PC_{\widehat{\mathcal{H}}^d}}(G)$ such that for the corresponding graph $\widehat{G}$, it holds that $\{u, v\} \in E(\widehat{G})$. Furthermore, there must exist a solution with corresponding graph $\widehat{G}$, such that for all twins $\{u, v\}$ of $G$ it holds that $\{u, v\} \in E(\widehat{G})$.*

*Proof.* Suppose we have an optimal solution $\{S_1, \ldots, S_k\}$ such that for $\widehat{G}$, defined as $\widehat{G} = G \widehat{\oplus} S_1 \widehat{\oplus} \ldots \widehat{\oplus} S_k$ it holds that $\{u, v\} \notin E(\widehat{G})$. Consider set of partial complements $\{S'_1, \ldots, S'_k\}$, defined as:

$$\begin{aligned} S'_i &= S_i && \text{if } (u \notin S_i \wedge v \notin S_i) \text{ or } (u \in S_i \wedge v \in S_i) \\ S'_i &= S_i \cup \{v\} && \text{if } u \in S_i \text{ and } v \notin S_i \\ S'_i &= S_i \setminus \{v\} && \text{if } v \in S_i \text{ and } u \notin S_i. \end{aligned}$$

We know that since $N_G(u) = N_G(v)$, and their neighbourhoods change in the same way, we find that

$$N_{(G \widehat{\oplus} S'_1 \widehat{\oplus} \ldots \widehat{\oplus} S'_k)}(v) = N_{(G \widehat{\oplus} S'_1 \widehat{\oplus} \ldots \widehat{\oplus} S'_k)}(u) = N_{\widehat{G}}(u) \cup \{v\}$$

Since $u \in N_{\widehat{G}}(u)$, we know that $\{u, v\} \in E(G \widehat{\oplus} S'_1 \widehat{\oplus} \ldots \widehat{\oplus} S'_k)$. We thus now that $u$ and $v$ end up in the same connected component, and $G \widehat{\oplus} S'_1 \widehat{\oplus} \ldots \widehat{\oplus} S'_k \in \widehat{\mathcal{H}}^d$. Therefore, $\{S'_1, \ldots, S'_k\}$ is an optimal solution that connects $u$ and $v$.

We could now iteratively repeat this process to ensure that all twins are connected, which gives us the second property of this lemma. ■

Thus, we only have to consider partitions of the vertices that put twin vertices into the same vertex subsets. This is sufficient to turn into an FPT algorthm:

**Algorithm *Looped_d-Clusterable*$(G, d, k)$**

1. **if** $\mathrm{rk}\,(G) > k + d$
2. **then return** FALSE
3. **else** Find the distinct neighbourhoods
4. Let $L$ list all vertex $d$-partitions, preserving vertices with the same neighbourhood.
5. BOOL=FALSE
6. **for** $(V_1, \ldots, V_d) \in L$
7. $r = \mathrm{rk}\,(G \widehat{\oplus} V_1 \widehat{\oplus} \cdots \widehat{\oplus} V_d)$
8. **if** $r < k$
9. **then** BOOL=TRUE
10. **if** $r = k$ and $\exists v \in V(G)$ such that $\{v, v\} \in E(G \widehat{\oplus} V_1 \widehat{\oplus} \cdots \widehat{\oplus} V_d)$
11. **then** BOOL=TRUE
12. **return** BOOL

**Lemma 4.14.** *Looped_d-Clusterable is correct and runs in time* $O\left(\frac{d^{2^{k+d}}}{d!} n^\omega\right)$*, where* $\omega < 2.373$ *is the matrix multiplication exponent.*

*Proof.* Observe that if the $\widehat{PC_{\widehat{\mathcal{H}}^d}}(G, k)$ is a yes-instance, we know that

$$\mathrm{rk}\,(A_G) \leq k + d.$$

This tells us that there are at most $2^{k+d}$ distinct neighbourhoods in $G$, as there are only $2^{k+d}$ subsets of $k + d$ rows. By Lemma 4.13, we know that there exists an optimal solution that puts vertices with the same neighbourhood in $G$ into the same connected component. Let $S_1, \ldots, S_{k'}$ be that solution, and let $V_1, \ldots, V_d$ be the connected components corresponding to this solution. Clearly, $S_1, \ldots, S_{k'}$ must be an optimal solution to $\widehat{PC_{\mathcal{G}}}(G \widehat{\oplus} V_1 \widehat{\oplus} \cdots \widehat{\oplus} V_d)$, where $\mathcal{G}$ is the family of edgeless graphs. Applying Theorem 4.1, we know that given $G \widehat{\oplus} V_1 \widehat{\oplus} \cdots \widehat{\oplus} V_d$, we can verify in time $O(n^\omega)$ whether such a solution exists. So by trying all possibilities for $V_1, \ldots, V_d$, we will find the correct one if it exists, and this will give us the desired answer.

Checking the rank takes $O(n^\omega)$. We can find the distinct neighbourhoods in time $O(n^2)$. There are $\frac{d^{2^{k+d}}}{d!}$ ways to partition the distinct neighbourhoods in $d$ sets. For each of these, we compute a new graph in time $O(n^2)$, and then compute a rank in time $O(n^\omega)$, and maybe check $n$ edges if the rank is exactly $k$. All together, this gives us the $O\left(\frac{d^{2^{k+d}}}{d!} n^\omega\right)$ bound. ■

An interesting question is whether or not this method will extend towards the graph family $\widehat{\mathcal{H}}$. It is clear that Lemma 4.13 extends to the general case, but since we do not know how many connected components the final graph will have, we cannot

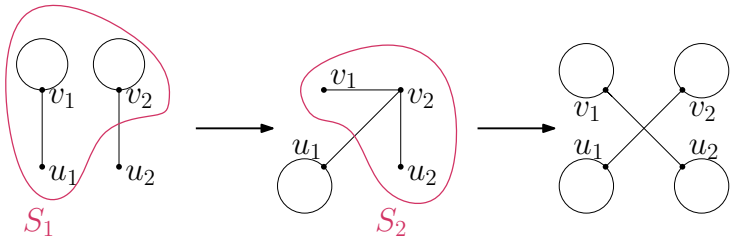


**Fig. 4.1:** Counter example. Even if two vertices share all their partial complements, they are not necessarily twins.

limit the number of distinct neighbourhoods based on the original rank. Therefore, we can't bound the number of distinct vertices by some function of $k$.

Another idea would be to see if vertices that are in the exact same sequence of partial complements have to be twin vertices. This does not turn out to be true, as we can see in Figure 4.1. Here, we see a unigraph $G$, and a solution $S_1, S_2$, such that $G \widehat{\oplus} S_1 \widehat{\oplus} S_2 \in \widehat{\mathcal{H}}$. Firstly, note that two partial complements is an optimal solution for this graph. Secondly, note that $v_1, v_2 \in S_1$ and $v_1, v_2 \in S_2$, but that $v_1$ and $v_2$ are not twin vertices. To conclude, this method gives us an FPT algorithm, but we cannot extend this algorithm to a more general case.

# 5 Implications for Simple Graphs

In the previous chapter we found results for unigraphs. In this chapter, we will use these results to find algorithms on simple graphs. Note that because we are studying simple graphs, the partial complement changes back to $\oplus$, as it was defined in Section 1.1. In other words, a partial complement no longer adds loops to vertices. This means we can't directly apply our results from Chapter 4. Therefore, we use the following approach; Given a problem instance on a simple graph, there must be some way to add loops to our instance graph, and some unigraph analogue of our simple graph goal family, that translates our problem into a problem we can solve by methods discussed in Chapter 4. First, we show that this method works. After that, we will look at the specific analogues of the problems discussed in Chapter 4, and see what happens in this new setting.

**Lemma 5.1.** *Let $\mathcal{F}$ be some family of graphs, and let $\widehat{\mathcal{F}}$ be a loop extension family of $\mathcal{F}$. Given a simple graph $G = (V, E)$, there exists a set $L \subseteq V(G)$ such that the unigraph $\widehat{G} = (V, E \cup \{\{u, u\} \mid \forall u \in L\})$ has the property*

$$|PC_{\mathcal{F}}(G)| = |\widehat{PC_{\widehat{\mathcal{F}}}}(\widehat{G})|$$

*Proof.* Let $k = |PC_{\mathcal{F}}(G)|$ and let $S_1, \ldots, S_k$ be an optimal solution for $PC_{\mathcal{F}}(G)$. Denote $G' = G \oplus S_1 \oplus \cdots \oplus S_k$. Thus, we know that $G' \in \mathcal{F}$. Let $\widehat{G}' \in \widehat{\mathcal{F}}$ be such that

$$(V(\widehat{G}'), E(\widehat{G}') \setminus \{\{u, u\} \mid \forall u \in V(\widehat{G}')\}) = G',$$

and let $L' \subseteq V(G')$ be such that

$$\widehat{G}' = (V(G'), E(G') \cup \{\{u, u\} \mid \forall u \in L'\}).$$

We construct $L$ in the following way. Given $v \in V(G)$, $v \in L$ if and only if $|\{i \mid v \in S_i\}|$ is odd and $v \notin L'$, or if $|\{i \mid v \in S_i\}|$ is even and $v \in L'$. By this construction we know that:

$$\widehat{G} \widehat{\oplus} S_1 \widehat{\oplus} \cdots \widehat{\oplus} S_k = \widehat{G}'.$$

Thus, $S_1, \ldots, S_k$ is also a solution for $\widehat{PC_{\widehat{\mathcal{F}}}}(\widehat{G})$. This allows us to conclude that $|PC_{\mathcal{F}}(G)| \geq |\widehat{PC_{\widehat{\mathcal{F}}}}(\widehat{G})|$. Furthermore, we know that $|PC_{\mathcal{F}}(G)| \leq |\widehat{PC_{\widehat{\mathcal{F}}}}(\widehat{G})|$, as any solution for $\widehat{PC_{\widehat{\mathcal{F}}}}(\widehat{G})$ must also be a solution for $PC_{\mathcal{F}}(G)$. This completes the proof. ■

So, if we can find $L$, i.e. if we find out how we need to add loops, we can translate problems on simple graphs to problems on unigraphs, which we know how to solve for certain graph families. Let us now look into specific graph families with loop family extensions that were discussed in Chapter 4.

## 5.1 Edgeless Graphs

In Section 4.1, we saw that in order to find how many partial complements are required to solve $\widehat{PC_{\mathcal{G}}}(G)$ (where $\mathcal{G}$ is the family of edgeless graphs), all we need to do is compute the rank of $A_G$, and look at its diagonal. In this section, we will use this as a base to create an $O(2^{2^k}n^\omega + n^3)$ time algorithm to solve $PC_{\mathcal{G}}(G, k)$.

We will formalize how the results from $\widehat{PC_{\mathcal{G}}}(\widehat{G})$ translate to $PC_{\mathcal{G}}(G)$. To achieve this we will introduce the rank minimizing diagonal addition problem. Furthermore, we will also look into critical cliques, and critical independent sets, and their relation with the rank minimizing diagonal addition problem. After we state our algorithm, we also show that 'repetitive' graphs can be solved more efficiently, for which we give an example. Let us start with the following definition.

**Definition.** Given a simple graph $G = (V, E)$, we define the *minimal looped rank of* $G$, denoted by $\eta_G$, as:

$$\eta_G = \min_{L \subseteq V} \{\mathrm{rk}\left(A_{\widehat{G}}\right) : \widehat{G} = (V, E \cup \{\{u, u\} \mid \forall u \in L\}), L \neq \emptyset\}$$

The following lemma provides the relevance of this concept in the case of edgeless graphs.

**Lemma 5.2.** *Let $\mathcal{G}$ be the family of edgeless graphs. Given a simple graph $G = (V, E)$ with $E \neq \emptyset$, it holds that*

$$|PC_{\mathcal{G}}(G)| = \eta_G$$

*Proof.* By Lemma 5.1 we know that there exists an $L^* \subseteq V(G)$ such that $|PC_{\mathcal{G}}(G)| = |\widehat{PC_{\mathcal{G}}}(G^*)|$, where

$$G^* = (V(G), E(G) \cup \{\{u, u\} \mid u \in L^*\}).$$

As $PC_{\mathcal{G}}(G)$ is a minimization problem, this relation clearly holds for the $L^* \subset V(G)$ for which $|\widehat{PC_{\mathcal{G}}}(G^*)|$ is minimal. We want to show that a graph corresponding to $\eta_G$, which we denote by $\widehat{G}$, will have the property $|\widehat{PC_{\mathcal{G}}}(G^*)| = |\widehat{PC_{\mathcal{G}}}(\widehat{G})|$. By Corollary 4.1.1, we know that $\eta_G = \mathrm{rk}\left(A_{\widehat{G}}\right) = |\widehat{PC_{\mathcal{G}}}(\widehat{G})|$, which together with our claim, would complete the proof.

If $L^* \neq \emptyset$, then we know that $\text{rk}\left(A_{G^*}\right) \geq \text{rk}\left(A_{\widehat{G}}\right)$. By Corollary 4.1.1, we know that $\text{rk}\left(A_{G^*}\right) = |\widehat{PC_{\mathcal{G}}}(G^*)|$, and $\text{rk}\left(A_{\widehat{G}}\right) = |\widehat{PC_{\mathcal{G}}}(\widehat{G})|$. By our pick of $L^*$, we know $|\widehat{PC_{\mathcal{G}}}(\widehat{G})| \geq |\widehat{PC_{\mathcal{G}}}(G^*)|$, and thus

$$|\widehat{PC_{\mathcal{G}}}(\widehat{G})| \geq |\widehat{PC_{\mathcal{G}}}(G^*)| = \text{rk}\left(A_{G^*}\right) \geq \text{rk}\left(A_{\widehat{G}}\right) = |\widehat{PC_{\mathcal{G}}}(\widehat{G})| = \eta_G,$$

meaning $|PC_{\mathcal{G}}(G)| = |\widehat{PC_{\mathcal{G}}}(G^*)| = \eta_G$.

If instead $L^* = \emptyset$, the proof is less obvious. Let $L' \subseteq V(G)$ be some vertex subset such that $|L'| = 1$, and let the unigraph $G'$ be defined as

$$G' = (V(G), E(G) \cup \{\{u, u\} \mid u \in L'\}).$$

As $A_{G^*}$ and $A_{G'}$ differ only in one row, we can extend the original row basis with this new row, which implies that

$$\text{rk}\left(A_{G'}\right) \leq \text{rk}\left(A_{G^*}\right) + 1$$

By our pick of $L^*$, we know that $|\widehat{PC_{\mathcal{G}}}(G^*)| \leq |\widehat{PC_{\mathcal{G}}}(G')|$. Applying Corollary 4.1.1, we find that:

$$|\widehat{PC_{\mathcal{G}}}(G')| = \text{rk}\left(A_{G'}\right) \leq \text{rk}\left(A_{G^*}\right) + 1 = |\widehat{PC_{\mathcal{G}}}(G^*)| \leq |\widehat{PC_{\mathcal{G}}}(G')|$$

Concluding that $|\widehat{PC_{\mathcal{G}}}(G^*)| = |\widehat{PC_{\mathcal{G}}}(G')|$. Using similar logic as before, we know that $\text{rk}\left(A_{G'}\right) \geq \text{rk}\left(A_{\widehat{G}}\right)$. By Corollary 4.1.1, we know that $\text{rk}\left(A_{G'}\right) = |\widehat{PC_{\mathcal{G}}}(G')|$, and $\text{rk}\left(A_{\widehat{G}}\right) = |\widehat{PC_{\mathcal{G}}}(\widehat{G})|$. By our pick of $L^*$, we know $|\widehat{PC_{\mathcal{G}}}(\widehat{G})| \geq |\widehat{PC_{\mathcal{G}}}(G^*)| = |\widehat{PC_{\mathcal{G}}}(G')|$, and thus

$$|\widehat{PC_{\mathcal{G}}}(\widehat{G})| \geq |\widehat{PC_{\mathcal{G}}}(G')| = \text{rk}\left(A_{G'}\right) \geq \text{rk}\left(A_{\widehat{G}}\right) = |\widehat{PC_{\mathcal{G}}}(\widehat{G})| = \eta_G,$$

meaning $|PC_{\mathcal{G}}(G)| = |\widehat{PC_{\mathcal{G}}}(G^*)| = |\widehat{PC_{\mathcal{G}}}(G')| = \eta_G$. This completes our proof. ■

Lemma 5.2 creates an interest in efficiently computing $\eta_G$. For clarity, we will state this as a formal problem on matrices instead of graphs.

**Rank Minimizing Diagonal Addition** (RMDA($A$)) Given a symmetric $GF[2]$ zero-diagonal $n \times n$ matrix $A$. For $L \subseteq \{1, \ldots, n\}$, we denote by $I_L$ the $n \times n$ matrix where $I_{L\{i,j\}} = 1$ if and only if $i = j$ and $i \in L$, and 0 otherwise. Find $L^* \subseteq \{1, \ldots, n\}$ with $L^* \neq \emptyset$ such that

$$\text{rk}\left(A + I_{L^*}\right) = \min_{L \subseteq \{1,\ldots,n\}} \{\text{rk}\left(A + I_L\right) \mid L \neq \emptyset\}$$

If we could efficiently solve RMDA($A_G$), we would also find the optimal solution for $PC_{\mathcal{G}}(G)$ by Lemma 5.2. For some non-trivial graphs $G$, we can find the optimal solution to RMDA($A_G$) relatively easily, by exploiting repetition in the graph. This will be shown in Lemma 5.7, where we show that $P_3^n$, the graph consisting of $n$ disjoint copies of $P_3$ can be turned into an edgeless graph with $2n$ partial complements. While these proofs for specific graphs are interesting, we want a way to solve RMDA($A_G$), and by extension $PC_{\mathcal{G}}(G)$ for any simple graph $G$. Consider the following definition.

**Definition.** A *critical clique* of a simple graph $G$ is a clique $K$ where for every vertex $v \in K$ we have that $N_G[v] = N_G[K]$, and $K$ is maximal under this property. Similarly, a *critical independent set* of a simple graph $G$ is an independent set $A$ where for every vertex $v \in A$ we have that $N_G(v) = N_G(A)$, and $A$ is maximal under this property.

Recall that two vertices $u$ and $v$ are called twin vertices if $N_G(u) = N_G(v)$ and $u \neq v$. We can imagine critical cliques and critical independent sets to be the simple graph parallel to sets of twin vertices in unigraphs. It is clear that if we add loops at all vertices of a critical clique, it will become a set of twin vertices, and adding loops at no vertices of a critical independent set also gives us a set of twin vertices. What is perhaps less obvious is the following:

**Lemma 5.3.** *Let $G$ be a simple graph. If $\widehat{G}$ is obtained from $G$ by adding loops at some vertices, and $u, v \in V(\widehat{G})$ are twins in $\widehat{G}$, then there exists a critical clique $K$ in $G$ or a critical independent set $A$ in $G$ such that $u, v \in K$ or $u, v \in A$.*

*Proof.* Since $u$ and $v$ are twins in $\widehat{G}$, we know that $N_{\widehat{G}}(u) = N_{\widehat{G}}(v)$. If $\{u, v\} \in E(G)$, then clearly $N_G[u] = N_G[v]$, and $u$ and $v$ are part of some critical clique $K$. Else, we know that $N_G(u) = N_G(v)$, and that $u$ and $v$ are part of some critical independent set. ■

In Section 4.2, we saw that we can use the rank of an adjacency matrix to bound the number of distinct neighbourhoods, i.e. the number of vertices of the graph after removing twin vertices. We want to make an analogous statement about critical cliques/independent sets.

**Lemma 5.4.** *Given a simple graph $G$. Let $\mathcal{K}$ be the family of all critical cliques $K$ of $G$ with $|K| \geq 2$, $\mathcal{A}$ be the family of all critical independent sets $A$ of $G$ with $|A| \geq 2$, and $\mathcal{S}$ be the family of one element subsets of $V(G)$, containing each vertex that is not covered by either $\mathcal{K}$ or $\mathcal{A}$ as a separate subset. We then know that every vertex $v$ appears in exactly one element of $\mathcal{K} \cup \mathcal{A} \cup \mathcal{S}$, and that $|\mathcal{K}| + |\mathcal{A}| + |\mathcal{S}| \leq 2^{\eta_G}$.*

*Proof.* First of, let us show that every vertex $v$ appears in exactly one element of $\mathcal{K} \cup \mathcal{A} \cup \mathcal{S}$. Consider $v \in V(G)$. Suppose there exists some $K \in \mathcal{K}$ such that $v \in K$. Furthermore, if there would exist another $K' \in \mathcal{K}$ such that $K \neq K'$ and $v \in K'$, then we would know that $N_G[K] = N_G[v] = N_G[K']$, but this would violate the maximality of critical cliques.

If there would exist an $A \in \mathcal{A}$ such that $v \in A$, let $w \in A$ with $w \neq v$, and $u \in K$ with $u \neq v$. We know that $u \neq w$, because $\{v, u\} \in E(G)$ and $\{v, w\} \notin E(G)$. Since $u \in N_G(v)$ and $N_G(w) = N_G(v)$, we know $u \in N_G(w)$. Since $w \in N_G[u] = N_G[v]$, we know $w \in N_G[v]$. But since $v, w \in A$, we know that $w \notin N_G[v]$. Thus, we can conclude that $v$ cannot be in both $K$ and $A$.

Now consider a vertex $u \in A$ for some $A \in \mathcal{A}$. If there exists an $A' \in \mathcal{A}$ with $A \neq A'$ and $u \in A'$, then this would again violate the maximality of critical independent sets. Lastly, if $\{w\} \in \mathcal{S}$, then by definition $w$ does not appear in any set in $\mathcal{K}$ or $\mathcal{A}$. Also, if $w$ does not appear in any set in $\mathcal{K} \cup \mathcal{A}$, then by definition $\{w\} \in \mathcal{S}$. Therefore, every vertex $v$ appears in exactly one element of $\mathcal{K} \cup \mathcal{A} \cup \mathcal{S}$.

It remains to show that $|\mathcal{K}| + |\mathcal{A}| + |\mathcal{S}| \leq 2^{\eta_G}$. Let $\widehat{G}$ be the unigraph corresponding to $\eta_G$. We know that $\operatorname{rk}\left(A_{\widehat{G}}\right) = \eta_G$, and that there are at most $2^{\operatorname{rk}\left(A_{\widehat{G}}\right)}$ distinct neighbourhoods in $\widehat{G}$. Thus, if we were to remove the twins in $\widehat{G}$, we would be left with at most $2^{\eta_G}$ vertices. By Lemma 5.3, a set of mutual twins in $\widehat{G}$ are all contained within some critical clique or critical independent set of $G$. This allows us to conclude that $|\mathcal{K}| + |\mathcal{A}| + |\mathcal{S}| \leq 2^{\eta_G}$. ■

The obvious next step is to look at how critical cliques and critical independent sets can be exploited to solve RMDA$(A_G)$.

**Lemma 5.5.** *Let $G$ be a simple graph, let $L^*$ be the vertex subset corresponding to an optimal solution to* RMDA$(A_G)$*, and let $X$ be a critical clique or critical independent set. It then holds that $X \subseteq L^*$ or $X \cap L^* = \emptyset$.*

*Proof.* Observe that if $|X| = 1$ the prove is immediate. So let us assume that $|X| \geq 2$. First of, suppose $X$ is a critical clique. Consider any subset $S \subseteq V(G)$ such that $X \subseteq S$, and consider the row space of $A_G + I_S$. As $X$ is a critical clique, any row that has a one in a column corresponding to some vertex of $X$ must have a one at all vertices of $X$. Therefore no row distinguishes between the vertices of $X$. Pick $v \in X$, let $S' = S - v$ and consider $A_G + I_{S'}$. We claim that

$$\operatorname{rk}\left(A_G + I_S\right) < \operatorname{rk}\left(A_G + I_{S'}\right).$$

This is because we know that there is a vector contained within the row space of $A_G + I_{S'}$ that distinguishes between the vertices of $X$. Since no row of $A_G + I_S$ has this distinction, we know that this new row can't be a linear combination of the old rows. Furthermore, since the complete row space of $A_G + I_S$ must be contained within $A_G + I_{S'}$, we can conclude that our claim is correct. We can repeat these arguments for strict subsets of $X$. We can't repeat it for removing $X$ completely, since we can then no longer claim that the complete row space of $A_G + I_S$ must be contained in $A_G + I_{S'}$. This tells us that a rank minimizing $S$ must either completely contain $X$, or be completely disjoint of $X$.

The proof for the case where $X$ is a critical independent set goes analogously by starting out with a subset $S \subseteq V(G)$ with $S \cap X = \emptyset$, and showing that adding a single vertex of $X$ to $S$ will result in increasing the rank by one. ■

This gives us enough insight to create an algorithm for $PC_{\mathcal{G}}(G, k)$, where $\mathcal{G}$ is the family of edgeless graphs.

**Algorithm *PC_to_Edgeless_no_Loops*($G, k$)**

1. Construct $\mathcal{K}$, $\mathcal{A}$ and $\mathcal{S}$
2. **if** $|\mathcal{K}| + |\mathcal{A}| + |\mathcal{S}| > 2^k$
3. **then return** FALSE
4. **else** bool=FALSE
5. **for** $L \subseteq (\mathcal{K} \cup \mathcal{A} \cup \mathcal{S}) \setminus \{\emptyset\}$
6. $G' = (V(G), E(G) \cap \{\{u, u\} \mid u \in U \text{ and } U \in L\})$
7. **if** $\mathrm{rk}\,(A_{G'}) \leq k$
8. **then** bool=TRUE
9. **return** bool

Here $\mathcal{K}, \mathcal{A}$ and $\mathcal{S}$ are as defined in Lemma 5.4. Let us prove the correctness of this algorithm:

**Lemma 5.6.** *PC_to_Edgeless_no_Loops is correct and runs in* $O(2^{2^k} n^\omega + n^3)$.

*Proof.* By Lemma 5.4, we know that $|\mathcal{K}| + |\mathcal{A}| + |\mathcal{S}| > 2^k$ implies that $k > \eta_G$. By Lemma 5.2, we know that $|PC_{\mathcal{G}}(G)| = \eta_G$. So if $|\mathcal{K}| + |\mathcal{A}| + |\mathcal{S}| > 2^k$, then we know that $|PC_{\mathcal{G}}(G)| > k$, and that $PC_{\mathcal{G}}(G, k)$ is a no-instance. By Lemma 5.5, we know that we only have to check the cases where either all vertices or no vertices of a critical clique or independent set are assigned loops. So that is what we do. Lastly, we know by Corollary 4.1.1 that if $\mathrm{rk}\,(A_{G'}) \leq k$, we have encountered a yes-instance. Note that we don't check the all $0$ diagonal option, as Lemma 5.2 tells us this is not necessary.

Let's compute the runtime. We can construct $\mathcal{K}, \mathcal{A}$, and $\mathcal{S}$ in time $O(n^3)$, as this is just a special way of comparing neighbourhoods of vertex pairs. Line 2 can be done in $O(n)$. If we go into the for loop of line 5, we will loop over at most $2^{2^k}$ cases. In each of these cases we create $G'$, which takes $O(n^2)$, and we compute a matrix rank in time $O(n^\omega)$, where omega is the matrix multiplication exponent. This gives us a total running time of $O(2^{2^k} n^\omega + n^3)$. ■

Thus, we have found an alternative algorithm to the algorithm theorised in Chapter 3. While we did not find the exact algorithm implied by Theorem 3.3, we know that algorithms based on Theorem 3.3 usually have a very large running time. Therefore, we assume that this alternative runs faster than the original. As promised, we will now look into a specific instance in which we can find a solution more efficiently. Namely, we consider $P_3^n$, the graph consisting of $n$ disjoint copies of $P_3$.

**Lemma 5.7.** *Let $\mathcal{G}$ be the family of edgeless graphs, and let $G = P_3^n$. Then $|PC_{\mathcal{G}}(G)| = 2n$.*

*Proof.* Let the vertices of $G$ be enumerated as shown in Figure 5.1. Consider the adjacency matrix $A_G$. It is a blockdiagonal matrix, where all the blocks $B_i$ are of the form

$$B_i = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Given some $L \subseteq V(G)$, let $x_j$ be an indicator of whether or not $j \in L$. Define block $B_i^*$ as

$$B_i^* = \begin{pmatrix} x_{3i-2} & 1 & 0 \\ 1 & x_{3i-1} & 1 \\ 0 & 1 & x_{3i} \end{pmatrix}$$

Let $\widehat{G}$ be a unigraph obtained by taking a copy of $G$, and adding loops at every vertex in $L$. Then $A_{\widehat{G}}$ is a block diagonal of the form

$$A_{\widehat{G}} = \begin{pmatrix} B_1^* & 0 & \cdots & 0 \\ 0 & B_2^* & & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & \cdots & B_n^* \end{pmatrix}$$

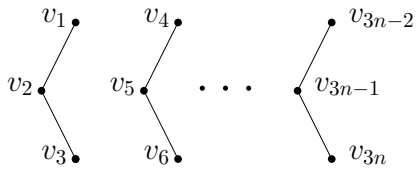


**Fig. 5.1:** Enumeration of the vertices in $P_3^n$

To find which $L$ minimizes $\mathrm{rk}\left(A_{\widehat{G}}\right)$, we must find out which $L$ minimizes $\mathrm{rk}\left(B_i^*\right)$ for each $i$. By trying all $8$ possibilities, we find that $\mathrm{rk}\left(B_i^*\right) \geq 2$, with equality if $x_{3i-2} = x_{3i} = 1$ and $x_{3i-1} = 0$. Therefore, by Lemma 5.2, we find that

$$|PC_{\mathcal{G}}(G)| = \min_{L \subseteq V(P_3^n)} \{\mathrm{rk}\left(A_{\widehat{G}}\right) \mid L \neq \emptyset\} = 2n.$$

This corresponds to our intuition. Since $G$ has $2n$ edges, we know that we can spend one partial complement per edge to achieve a solution to $PC_{\mathcal{G}}(G)$ of size $2n$, but now we know that we cannot do any better. ■

## 5.2 $P_3$-Free Graphs of at most $d$ Connected Components

In this section we will look at the graph family $\mathcal{H}^d$, the family of $P_3$-free graphs of at most $d$ connected components. In Section 4.2, we saw an $O(\frac{d^{2^{k+d}}}{d!}n^\omega)$ algorithm for $\widehat{PC_{\widehat{\mathcal{H}}^d}}(G,k)$, where $\widehat{\mathcal{H}}^d$ is the family of $P_3^*$-free reflexive unigraphs. Over the course of this section, we extend the algorithm from Section 4.2 to simple graphs. We achieve this in a way that is very similar to Section 5.1, but the argumentation is different. As in the Section 5.1, we look into critical cliques and critical independent sets. We are again able to show that an optimal addition of loops either puts loops on all or on non of the vertices of a critical clique/independent set. By Lemma 5.4, we know the amount of critical cliques, critical independent sets, and single vertices is bounded by the rank, which is in turn bounded by our parameters $d$ and $k$. This allows us to create an algorithm for $PC_{\mathcal{H}^d}(G,k)$ that runs in time $O(\frac{(2d)^{2^{k+d}}}{d!}n^\omega + n^3)$, by trying all loop assignments that adhere to the property on critical cliques and independent sets. We shall start by showing that the rank of a yes-instance is bounded by a function of the parameters:

**Lemma 5.8.** *Let $\mathcal{H}^d$ be the family of $P_3$-free graphs consisting of at most $d$ connected components. Given a simple graph $G = (V, E)$, it holds that*

$$|PC_{\mathcal{H}^d}(G)| + d \geq \eta_G.$$

*Proof.* Suppose we knew the optimal solution $PC_{\mathcal{H}^d}(G)$, and suppose the connected components of the solution graph are $C_1, \ldots, C_d$ (some of which can be empty). Let

$$G' = G \oplus C_1 \oplus \cdots \oplus C_d$$

Then we know that $|PC_{\mathcal{G}}(G')| = |PC_{\mathcal{H}^d}(G)|$. By Lemma 5.2, we know that

$$|PC_{\mathcal{G}}(G')| = \eta_{G'}.$$

As $G'$ is created from $G$ and taking $d$ partial complements, we know that if the unigraph $\widehat{G}'$ corresponds to $\eta_{G'}$, then $\widehat{G}'\widehat{\oplus}C_1\widehat{\oplus}\cdots\widehat{\oplus}C_d$ gives us a valid loop assignment for $G$. Therefore, $\eta_G \leq \eta_{G'} + d$. Combining these results we find that

$$|PC_{\mathcal{H}^d}(G)| + d = |PC_{\mathcal{G}}(G')| + d = \eta_{G'} + d \geq \eta_G.$$

This completes the proof. ■

**Corollary 5.8.1.** *If $PC_{\mathcal{H}^d}(G, k)$ is a yes-instance, then $k + d \geq \eta_G$.*

*Proof.* If $PC_{\mathcal{H}^d}(G, k)$ is a yes-instance, we know that $|PC_{\mathcal{H}^d}(G)| \leq k$. Combining this with the results of Lemma 5.8 shows that $k + d \geq \eta_G$. ■

Now, let us show the desired property of critical cliques and critical independent sets. In Lemma 4.13, we saw that there exists a solution in which a pair of twin vertices ends up in the same connected component. For critical cliques and critical independent sets, this is not guaranteed. However, we are only interested in how we need to add loops to the vertices of a critical clique/independent set. We know that in our goal family of unigraphs, every vertex should have a loop. Thus, we need to assign loops to our original simple graph such that vertices that appear in an even amount of partial complements start with a loop, whereas vertices that appear in an odd amount of partial complements start without a loop. Combining this notion of parity with the similarity of vertices within critical cliques and critical independent sets leads us to the following lemma.

**Lemma 5.9.** *Let $G$ be an instance of $PC_{\mathcal{H}^d}$, and let $\mathcal{K}$ and $\mathcal{A}$ be the critical cliques and critical independent sets of $G$ of size larger than 1. For a solution $S_1, \ldots, S_k$, we define $L \subseteq V(G)$ as*

$$L = \{v \mid v \in V(G) \text{ for which } |\{i \mid v \in S_i\}| \text{ is even}\}$$

*There exists an optimal solution $S_1, \ldots, S_k$ and corresponding vertex subset $L$ such that $\forall X \in \mathcal{K} \cup \mathcal{A}$, it holds that $X \subseteq L$ or $X \cap L = \emptyset$.*

*Proof.* Let $S_1, \ldots, S_k$ be an optimal solution to $PC_{\mathcal{H}^d}(G)$. Assume that $S_1, \ldots, S_k$ does not satisfy the condition, because else we would be done. So, there exists an $X \in \mathcal{K} \cup \mathcal{A}$ such that there exists $u, v \in X$ with $u \in L$ but $v \notin L$. We know $X$ is either a critical clique or a critical independent set, and we will handle these cases separately.

First, suppose $X$ is a critical clique. We define a new set of partial complements $S'_1, \ldots, S'_k$ as follows:

$$\begin{aligned} S'_i &= S_i \quad \text{if } (u \notin S_i \wedge v \notin S_i) \text{ or } (u \in S_i \wedge v \in S_i) \\ S'_i &= S_i \cup \{v\} \quad \text{if } u \in S_i \text{ and } v \notin S_i \\ S'_i &= S_i \setminus \{v\} \quad \text{if } v \in S_i \text{ and } u \notin S_i. \end{aligned}$$

This new set of partial complements gives us a new $L'$. Note that since $u \in L$, and we only change which partial complements $v$ is in, $u \in L'$. Also note that $u \in S'_i$ if and only if $v \in S'_i$. This implies that $v \in L'$. Denote

$$G' = (V(G), E(G) \cup \{\{w, w\} \mid w \in L'\}).$$

We know that since $N_{G'}(v) = N_{G'}(u)$, and their neighbourhoods change in the same way, we find that

$$N_{(G' \widehat{\oplus} S'_1 \widehat{\oplus} \ldots \widehat{\oplus} S'_k)}(v) = N_{(G' \widehat{\oplus} S'_1 \widehat{\oplus} \ldots \widehat{\oplus} S'_k)}(u) = N_{(G \oplus S_1 \oplus \cdots \oplus S_k)}(u) \cup \{u, v\}.$$

Since the connected component containing $u$ in $G \oplus S_1 \oplus \cdots \oplus S_k$ is $P_3$-free, we know that the component containing $u$ in $G \oplus S'_1 \oplus \ldots \oplus S'_k$ is also $P_3$-free; It consists of the same vertices plus $v$, $u$ and $v$ are adjacent, and every neighbour of $u$ is a neighbour of $v$.

The other components also remain $P_3$-free, as a $P_3$-free graph does not lose its $P_3$-free property when a vertex is removed. Obviously, we don't increase the number of connected components. So we can conclude that $G \oplus S'_1 \oplus \ldots \oplus S'_k \in \mathcal{H}^d$, and thus that $S'_1, \ldots, S'_k$ is a correct solution, of optimal size. We can repeat this trick until all of the vertices of $X$ are in $L'$.

If instead $X$ is a critical independent set, our proof is almost identical. We define a new set of partial complements $S'_1, \ldots, S'_k$ as follows:

$$\begin{aligned} S'_i &= S_i \quad \text{if } (u \notin s_i \wedge v \notin S_i) \text{ or } (u \in S_i \wedge v \in S_i) \\ S'_i &= S_i \setminus \{u\} \quad \text{if } u \in S_i \text{ and } v \notin S_i \\ S'_i &= S_i \cup \{u\} \quad \text{if } v \in S_i \text{ and } u \notin S_i. \end{aligned}$$

This new set of partial complements gives us a new $L'$. Note that since $v \notin L$, and we only change which partial complements $u$ is in, $v \notin L'$. Also note that $u \in S'_i$ if and only if $v \in S'_i$. This implies that $u \notin L'$. Denote

$$G' = (V(G), E(G) \cup \{\{w, w\} \mid w \in L'\}).$$

We know that since $N_{G'}(v) = N_{G'}(u)$, and their neighbourhoods change in the same way, we find that

$$N_{(G' \widehat{\oplus} S'_1 \widehat{\oplus} \ldots \widehat{\oplus} S'_k)}(v) = N_{(G' \widehat{\oplus} S'_1 \widehat{\oplus} \ldots \widehat{\oplus} S'_k)}(u) = N_{(G \oplus S_1 \oplus \cdots \oplus S_k)}(v) \cup \{u, v\}.$$

Since the connected component containing $v$ in $G \oplus S_1 \oplus \cdots \oplus S_k$ is $P_3$-free, this shows that the component containing $v$ in $G \oplus S'_1 \oplus \ldots \oplus S'_k$ is also $P_3$-free; It consists of the same vertices plus $u$, $u$ and $v$ are adjacent, and every neighbour of $v$ is a neighbour of $u$.

The other components also remain $P_3$-free, as a $P_3$-free graph does not lose its $P_3$-free property when a vertex is removed. Obviously, we don't increase the number of connected components. So we can conclude that $G \oplus S'_1 \oplus \ldots \oplus S'_k \in \mathcal{H}^d$, and thus that $S'_1, \ldots, S'_k$ is a correct solution, of optimal size. We can repeat this until no vertices of $X$ are in $L'$.

Since the alternate solutions only influence vertices in one particular critical clique or critical independent set $X$, we can repeat this trick for any $X \in \mathcal{K} \cup \mathcal{A}$. This will give us a solution that fits the conditions of the lemma. ■

Combining this result with results from previous sections, we give the following FPT algorithm.

**Algorithm *Loopless_d-Clusterable*(**$G, d, k$**)**

```
1.  Construct K, A and S
2.  if |K| + |A| + |S| > 2^{k+d}
3.     then return FALSE
4.     else  BOOL=FALSE
5.           for L ⊆ (K ∪ A ∪ S)
6.                 G' = (V(G), E(G) ∩ {{u, u} | u ∈ U and U ∈ L})
7.                 if Looped_d-Clusterable(G', d, k)
8.                    then BOOL=TRUE
9.           return BOOL
```

**Lemma 5.10.** *Loopless_d-Clusterable is correct and runs in time* $O(\frac{(2d)^{2^{k+d}}}{d!}n^\omega + n^3)$.

*Proof.* Suppose $|\mathcal{K}| + |\mathcal{A}| + |\mathcal{S}| > 2^{k+d}$. By Lemma 5.4, this implies that $k + d < \eta_G$. Corollary 5.8.1 tells us that this implies that $PC_{\mathcal{H}^d}(G, k)$ is a no-instance.

If $|\mathcal{K}| + |\mathcal{A}| + |\mathcal{S}| \leq 2^{k+d}$, we might be dealing with a yes-instance. By Lemma 5.1, we know that there is a way to assign loops to the vertices of $G$ to create $G'$ such that $PC_{\mathcal{H}^d}(G) = \widehat{PC_{\widehat{\mathcal{H}}^d}}(G')$. By Lemma 5.9, we know that there is an optimal loop assignment that assigns loops to exactly all or non of the vertices within any critical clique and independent sets. The algorithm goes over all $2^{2^{k+d}}$ loop-assignments of that form, and computes $\widehat{PC_{\widehat{\mathcal{H}}^d}}(G', k)$ with *Looped_d-Clusterable*$(G', d, k)$, which is proven correct in Lemma 4.14. This proves the correctness of this algorithm.

For the running time: We can find $\mathcal{K}, \mathcal{A}$, and $\mathcal{S}$ in time $O(n^3)$. After that, we do a verification in line 2, which can be done in time $O(2^{k+d})$. Then, we dive into a loop of $2^{2^{k+d}}$ instances. In each of these, we create a unigraph version of our original simple graph, which we will bound on time $O(n^2)$. After that, we compute *Looped_d-Clusterable*, which can be done in time $O(\frac{d^{2^k+d}}{d}n^\omega)$. Combining all of this gives us the desired runtime. ■

So we see that we can adapt the algorithm from Section 4.2 for the case where graphs don't have loops, given us another alternative to the algorithms theorised in Chapter 3.

# 6 Conclusion and Discussion

Over the course of this thesis, we have studied $PC_{\mathcal{F}}(G)$ for various graph families $\mathcal{F}$. We have found that for graph families with bounded rank-width, for which $PC_{\mathcal{F}}(G)$ is expressible in $\text{MSO}_1$ logic, $PC_{\mathcal{F}}(G,k)$ is FPT with regards to $k$. We have seen that the families of $H$-free simple graphs are included within these constraints, if and only if $H$ is an induced subgraph of $P_4$. For these families, we know that $PC_{\mathcal{F}}(G,k)$ permits an FPT algorithm.

Furthermore, we have created an $O(2^{2^k}n^\omega + n^3)$ time algorithm for $PC_{\mathcal{G}}(G)$, where $G$ is the family of $P_2$-free simple graphs. This was achieved by restricting the amount of critical cliques and critical independent sets, and translating our instance to a unigraph instance. In a similar manner we found an $O(\frac{(2d)^{2^{k+d}}}{d!}n^\omega + n^3)$ time algorithm for $PC_{\mathcal{H}^d}(G)$, where $\mathcal{H}^d$ is the family of $P_3$-free simple graphs of at most $d$ components.

Moreover, we saw that unigraphs were vital in discovering these algorithms. By looking into unigraphs, we found analogues for our problems in the case that loops are permitted. As this implies that taking a partial complement would also add or remove any loops on the set of vertices we are taking a partial complement on, suddenly we found a good way to represent the change a partial complement makes to an adjacency matrix. Namely, we know that this can be represent as the addition of a symmetric $GF[2]$ rank one matrix. This allowed us to introduce the notion of symmetric rank, and symmetric rank decompositions. We were able to say a lot about symmetric rank, and were eventually able to show that the symmetric rank of a matrix $A$ is equal to the regular rank of $A$ if $A$ contains at least one non-zero diagonal entry, whereas if $A$ has a diagonal completely zero but is not the complete zero matrix, its symmetric rank must be exactly one more than the regular rank. This gave us an $O(n^\omega)$ time algorithm to solve $|\widehat{PC_{\mathcal{G}}}(G)|$. Combined with a bound on the number of distinct neighbourhoods, we also used this notion to find a $O\left(\frac{d^{2^{k+d}}}{d!}n^\omega\right)$ time algorithm for $\widehat{PC_{\mathcal{H}^d}}(G,k)$.

To conclude, we have a set of criteria for a graph family $\mathcal{F}$ so that $PC_{\mathcal{F}}(G,k)$ is FPT, a set of examples of families fitting these criteria, and some other algorithms for specific graph families. To this extend, we have reached the goal of our project.

Furthermore, the introduction of symmetric rank, along with all the results we have generated on this notion are nice additional achievements.

On the other hand, though we achieved our goal, there is one vital criteria we did not achieve. Namely, we have not been able to prove the NP-completeness of $PC_{\mathcal{F}}(G, k)$ for any of the graph families $\mathcal{F}$ that we studied in detail. This is unfortunate. If it were to turn out that $PC_{\mathcal{F}}(G, k)$ is solvable in polynomial time for any of the graph families we have studied, then the observation that they allow an FPT algorithm parametrized by $k$ becomes obsolete. If we were able to prove NP-completeness, this would be less likely to happen. However, we were simply not able to determine whether our problems were NP-complete or not. In Chapter 7, we will go into more detail about our attempts at proofs, and how one might go about such a proof. We expect that $PC_{\mathcal{F}}(G, k)$ is indeed NP-complete for all families $\mathcal{F}$ that we have studied in detail within this thesis. Also, recall that $PC_{\mathcal{F}}(G, 1)$ is known to NP-complete if $\mathcal{F}$ is the family of $r$-regular graphs [14], thus we know that our problem will be NP-complete for some graph families.

It is important to note that though we were not able to guarantee NP-completeness, our results remain interesting. Even if eventually it turns that our problems are solvable in polynomial time, the methods we used to find our FPT algorithms still might give us insight into how to improve our running times, as we've shown bounds on distinct neighbourhoods, and shown that we can treat vertices with the same neighbourhood in the same manner. This could still be useful to decrease the running time of a polynomial time algorithm. Furthermore, all our results on symmetric rank are interesting regardless of their implementation in our algorithms for $PC_{\mathcal{F}}(G, k)$. Therefore, these results will keep their worth even if $PC_{\mathcal{F}}(G, k)$ is polynomial time solvable for some of the graph families we have studied. All in all, the results produced over the course of this project are relevant, even without the guarantees of NP-completeness.

Another point worth mentioning is that we might have been too quick to write off an approach on clique-width or rank-width based on Theorem 3.4, because the upperbounds for the dependency on $k$ used by Courcelle, Makowsky, and Rotics are incredibly large. Whilst this is most certainly true, in applications of this theorem lower upperbounds are achieved using problem specific properties. So, there might be an algorithm exploiting the bounded rank-width or bounded clique-width that we have overlooked.

Let us consider how such an algorithm would be constructed, starting with an algorithm using bounded clique-width. Most based on bounded clique-width use dynamic programming with the $k'$-expression of the input graph [16]. We know that disjoint unions of two graphs are almost certainly part of this $k'$-expression. This

puts us in a difficult situation. One might assume that if $G = G_1 \uplus G_2$, it holds that $|PC_{\mathcal{F}}(G)| = |PC_{\mathcal{F}}(G_1)| + |PC_{\mathcal{F}}(G_2)|$ for some of the graph families that we discuss, but a proof for this is not obvious. In fact, on unigraphs, we know that this does not hold. Let $G = P_2 \uplus P_2$, (so no vertices have loops). Using Theorem 4.1, we find that $|\widehat{PC_{\mathcal{G}}}(G)| = 5 < 3 + 3 = |\widehat{PC_{\mathcal{G}}}(P_2)| + |\widehat{PC_{\mathcal{G}}}(P_2)|$. Of course, this does not rule out that it holds for simple graphs. As we will show in Chapter 7, we do have a method of proving or disproving this fact for graph family $\mathcal{G}$ in mind. However, it is based on the results we achieved whilst pursuing the approach that is not based on clique-width or rank-width. Since we could not use this method without pursuing the other approach, and did not think of other methods, we refrained from focussing on algorithms based on bounded clique-width.

Instead, we might have based an algorithm on bounded rank-width. As every partial complement can decrease the rank-width by at most one, and the goal families we consider have low rank-width, we might consider trying to decrease the rank-width with every partial complement. If we have a rank-decomposition, we can tell which cuts are of the highest rank, and by analysing this cut matrix, we can find a partial complement to decrease the rank of this particular cut. However, we run into several problems here. First of, though we are certain that we decrease the rank of this particular cut, it might be the case that the other edges we create (which are not crossing the cut) increase the rank of other cuts in our rank-decomposition, thus restricting us from decreasing the rank. Furthermore, trying all the cuts of maximal rank will not give you an obvious guarantee that you will find a partial complement that decreases the rank if it exists. One could imagine that there are two distinct rank-decompositions of the same graph with minimal width, where one contains a cut hinting at the required partial complement, and the other does not contain this cut. Excluding this possibility is not obvious, and it is also not obvious to show how to work around this problem if it does exist. This is why we refrained from focusing on algorithms based on bounded rank-width.

# 7 Open Questions

Over the course of this project, we ran into some problems that we were not able to solve. In this chapter, you will find a list of those problems, combined with a list of problems that we deemed to be interesting for further research.

### Are our problems NP-Hard?

We still need to prove NP-hardness for any variants of graph editing by partial complements that we studied in detail. We currently don't really have a plan for how we can do this. We seemed to have a reduction from set cover to $PC_{\mathcal{H}}(G)$, where $\mathcal{H}$ is the family $P_3$-free simple graphs. However, a key part in this proof turned out to be false, and this mistake did not seem to be fixable by some clever gadgetering.

Another idea is to prove the NP-hardness of Rank Minimizing Diagonal Addition. This surely seems to be NP-hard, as it is somewhat related to the Matrix Rigidity problems. If this is in fact provably NP-hard, we can show that solving $PC_{\mathcal{G}}$, where $\mathcal{G}$ is the family of edgeless graphs, translates to a solution of RMDA, meaning it must also be NP-hard. From here, we might be able to prove NP-hardness of other variants, as $PC_{\mathcal{G}}$ seems to be the easiest variant of loopless problems.

It is unfortunate that we were not able to prove NP-hardness for any variant of our problem that we studied in detail. Especially since if it turns out that our problems are in fact solvable in polynomial time, our FPT results are obsolete.

### How do we solve $\widehat{\mathbf{PC}}_{\widehat{\mathcal{H}}}(G,k)$ efficiently?

We saw in Section 4.2 that we can solve $\widehat{PC_{\widehat{\mathcal{H}}^d}}(G,k)$. We want to find a way to extend this solution, or find an entirely new solution, for $\widehat{PC_{\widehat{\mathcal{H}}}}(G,k)$. This has proven to be hard, as the most of our results are based on rank, and we cannot say much about the rank of unigraphs in $\widehat{\mathcal{H}}$. This also means that we cannot bound the number of vertices of $G \in \widehat{\mathcal{H}}$ by a function of $|\widehat{PC_{\widehat{\mathcal{H}}}}(G)|$. So it will be hard to find an efficient algorithm for this. We do know that this family of graphs satisfies the conditions of Theorem 3.2, implying that the problem is FPT. But the question

remains whether or not we can do something more efficient than the algorithm implied by Theorem 3.2. As our original interest was sparked from the cluster editing problem, it seems especially interesting to see if we can find an efficient way to solve this graph family.

### Can we simplify the proof of Theorem 4.1?

Currently, the proof of Theorem 4.1 is based on a lot of small lemma's and algorithms, meaning that it takes a long time to prove this theorem. It might be interesting to know whether there exists a shorter, and simpler proof.

### Can we improve Theorem 4.12?

Let us restate this theorem for clarity:

**Theorem 4.12.** *Let $A$ be a symmetric $GF[2]$ matrix, with at least one non-zero diagonal entry. Denote $\mathrm{rk}(A) = r$, and let $u_1, \ldots, u_r$ be a symmetric rank decomposition of $A$. Given some $GF[2]$ vector $v$, it holds that $\mathrm{rk}\left(A + vv^T\right) = r - 1$, then there exists an $I \subseteq \{1, \ldots, r\}$ such that*

$$v = \sum_{i \in I} u_i.$$

During experimentation, it appeared to be the case that $|I|$ was necessarily odd, and that for any $I$ with $|I|$ odd the corresponding $v$ would decrease the rank. This is a much stronger result than the version of this theorem we have now. However, when we tried to prove this, we could not find any arguments that would suggest that this holds. Though we can't find arguments to base this claim on, it is an interesting concept, and it does not seem to be an absurd claim.

Furthermore, we want to obtain a similar result for the case where $A$ has an all zero diagonal. Of course, we know by Lemma 4.7 that no $v$'s will decrease the rank. However, we might investigate what $v$'s decrease the symmetric rank, as all of these (non-zero) $v$'s are part of an optimal symmetric rank decomposition for $A$.

### How can we deal with disjoint unions?

In order to create a clique-width based approach, we will need to know how to deal with graphs that are disjoint unions of two graphs. Of course, interest in this is not

restricted to a clique-width based approach, but it is a good source of motivation. So, the obvious question becomes, for what graph families $\mathcal{F}$ does the following relation hold:

$$|PC_{\mathcal{F}}(G_1 \uplus G_2)| = |PC_{\mathcal{F}}(G_1)| + |PC_{\mathcal{F}}(G_2)|$$

We have reason to believe that this is linked to the following question.

### Do there exist a simple graphs $G \notin \mathcal{G}$ such that rk$(A_G) < \eta_G$?

The reason we think these two concepts are linked is because a graph $G \notin \mathcal{G}$ for which rk$(A_G) < \eta_G$ would imply that $|PC_{\mathcal{G}}(G \uplus G)| = 2 \cdot |PC_{\mathcal{G}}(G)| - 1$. Furthermore, if for all simple graphs $G$ it holds that $\mathrm{rk}\,(A_G) \geq \eta_G$, then using Lemma 5.2 we can prove that

$$|PC_{\mathcal{G}}(G_1 \uplus G_2)| = \eta_{(G_1 \uplus G_2)} = \eta_{G_1} + \eta_{G_2} = |PC_{\mathcal{G}}(G_1)| + |PC_{\mathcal{G}}(G_2)|$$

for all simple graphs $G_1$ and $G_2$. Thus these two concepts form an if and only if relation for the family of edgeless graphs. Furthermore, deciding whether there exists a graph $G \notin \mathcal{G}$ such that $\mathrm{rk}\,(A_G) < \eta_G$ seems more approachable then deciding whether $|PC_{\mathcal{G}}(G_1 \uplus G_2)| = |PC_{\mathcal{G}}(G_1)| + |PC_{\mathcal{G}}(G_2)|$ for all simple graphs $G_1$ and $G_2$.

### What happens to symmetric rank over different fields?

In our project we introduced the notion of symmetric rank. We saw that over $GF[2]$, the symmetric rank is equal to the regular rank, unless the matrix has complete zero diagonal. Over the real numbers, we know that

$$xy^T + yx^T = \frac{1}{2}\left((x+y)(x+y)^T - (x-y)(x-y)^T\right),$$

which, combined with a slightly altered version of *Sym_Rank_Base*, tells us that the symmetric rank is equal to the regular rank [15]. But what about other fields? What can we say about symmetric rank in $GF[3]$? Or generally, what can we say about symmetric rank in $GF[p]$ for $p$ some prime number?

### Can we find more applications of symmetric rank?

In our problem, symmetric rank was introduced because it directly tells us something about $\widehat{PC_{\mathcal{G}}}$. However, it is unclear if there are more problems for which we can use

symmetric rank. Furthermore, if we extend the notion of symmetric rank to other finite fields, are there any more applications to be found?

# Bibliography


[1] Sebastian Böcker. A golden ratio parameterized algorithm for cluster editing. In Costas S. Iliopoulos and William F. Smyth, editors, *Combinatorial Algorithms*, pages 85–95, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[2] André Bouchet. Recognizing locally equivalent graphs. *Discrete Mathematics*, 114(1):75 – 86, 1993.

[3] James R. Bunch and John E. Hopcroft. Triangular factorization and inversion by fast matrix multiplication. *Mathematics of Computation*, 28(125):231–236, 1974.

[4] Jianer Chen and Jie Meng. A 2k kernel for the cluster editing problem. In My T. Thai and Sartaj Sahni, editors, *Computing and Combinatorics*, pages 459–468, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[5] Ho Yee Cheung, Tsz Chiu Kwok, and Lap Chi Lau. Fast matrix rank algorithms and applications. *J. ACM*, 60(5):31:1–31:25, October 2013.

[6] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[7] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[8] Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33(2):125–150, Apr 2000.

[9] Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1):77 – 114, 2000.

[10] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2015.

[11] Konrad K. Dabrowski and Daniël Paulusma. Clique-width of graph classes defined by two forbidden induced subgraphs†. *The Computer Journal*, 59(5):650–666, 2016.

[12] Andrzej Ehrenfeucht, Jurriaan Hage, Tero Harju, and Grzegorz Rozenberg. Complexity issues in switching of graphs. In *Theory and Application of Graph Transformations*, pages 59–70, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[13] Jelínková Eva and Kratochvíl Jan. On switching to $H$-free graphs. *Journal of Graph Theory*, 75(4):387–405, 2013.

[14] Fedor V. Fomin, Petr A. Golovach, Torstein J.F. Strømme, and Dimitrios M. Thilikos. Partial complementation of graphs. In *16th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2018)*, Leibniz International Proceedings in Informatics (LIPIcs), pages 21:1–21:13, 2018.

[15] hardmath. Writing real symmetric matrices as linear combination of rank one symmetric terms $uu^t$. Computational Science Stack Exchange. https://scicomp.stackexchange.com/q/7832 (version: 2013-06-28).

[16] Petr Hliněný, Sang-il Oum, Detlef Seese, and Georg Gottlob. Width parameters beyond tree-width and their applications. *Comput. J.*, 51(3):326–362, May 2008.

[17] Oscar H Ibarra, Shlomo Moran, and Roger Hui. A generalization of the fast LUP matrix decomposition algorithm and applications. *Journal of Algorithms*, 3(1):45 – 56, 1982.

[18] Marcin Kamiński, Vadim V. Lozin, and Martin Milanič. Recent developments on graphs of bounded clique-width. *Discrete Applied Mathematics*, 157(12):2747 – 2761, 2009. Second Workshop on Graph Classes, Optimization, and Width Parameters.

[19] Richard M. Karp. In *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations*, pages 85–103, Boston, MA, 1972. Springer US.

[20] Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, Sep 2012.

[21] François Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, ISSAC '14, pages 296–303, New York, NY, USA, 2014. ACM.

[22] Sang-il Oum. Rank-width and vertex-minors. *Journal of Combinatorial Theory, Series B*, 95(1):79 – 100, 2005.

[23] Sang-il Oum. Approximating rank-width and clique-width quickly. *ACM Trans. Algorithms*, 5(1):10:1–10:20, December 2008.

[24] Sang-il Oum. Rank-width: Algorithmic and structural results. *Discrete Applied Mathematics*, 231:15 – 24, 2017. Algorithmic Graph Theory on the Adriatic Coast.

[25] Sang-il Oum and Paul Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B*, 96(4):514 – 528, 2006.

[26] Johan J. Seidel. Graphs and two-graphs. In *Proceedings 5th Southeastern Conference on Combinatorics, Graph Theory and Computing (Boca Raton FL, USA, 1974)*, pages 125–143, 1974.

[27] Arne Storjohann. Integer matrix rank certification. In *Proceedings of the 2009 International Symposium on Symbolic and Algebraic Computation*, ISSAC '09, pages 333–340, New York, NY, USA, 2009. ACM.