Vertex Partitioning Problems: Characterization, Complexity and Algorithms on Partial k-Trees

Jan Arne Telle

CIS-TR–94-18
June 1994

Department of Computer and Information Science
University of Oregon

Abstract

This thesis investigates the computational complexity of algorithmic problems defined on graphs. At the abstract level of the complexity spectrum we discriminate polynomial-time solvable problems from $\mathcal{NP}$-complete problems, while at the concrete level we improve on polynomial-time algorithms for generally hard problems restricted to tree-decomposable graphs.

One contribution of this thesis is a precise characterization of vertex partitioning problems which include variants of domination, coloring and packing. An elaboration of this characterization is given for problems defined over vertex subsets and over maximal/minimal vertex subsets. We introduce several new graph parameters as vertex partition generalizations of classical parameters. The given characterizations provide a basis for a taxonomy of vertex partitioning problems, facilitating their common algorithmic treatment and allowing for their uniform complexity classification.

We explore the computational complexity of two important types of problems within this taxonomy: vertex subset optimization problems and H-covering problems. The taxonomy is particularly useful in categorizing and analyzing the complexity of vertex subset problems, of which there are a great variety. Our investigation of the complexity of vertex subset problems uncovers several infinite classes of $\mathcal{NP}$-complete and of polynomial-time solvable problems. These results are contrasted and compared with the complexity of classical vertex subset problems. We also develop a methodology useful in analyzing the complexity of H-covering, a problem parameterized by a fixed graph H. As an illustration, we settle the complexity of the H-covering problem for any simple graph H on at most 6 vertices. We design efficient algorithms for H-covering problems by reduction to the 2-SAT problem and by reduction to factorization problems in regular graphs.

Another contribution of this thesis is a methodology for the design of practical algorithms for generally $\mathcal{NP}$-hard problems restricted to partial k-trees. Based on very simple graph operations, we define a binary parse tree of partial k-trees that facilitates algorithm derivation. We account for dependency on the treewidth k in analysis of the computational complexity of the resulting algorithms.

These contributions culminate in applying the partial k-tree algorithm methodology to the general class of vertex partitioning problems. The input graph in the resulting algorithms is assumed to be given with a width k tree-decomposition, and the answer is computed by a dynamic programming bottom-up traversal of its binary parse tree. We give the first algorithms for these problems with reasonable time complexity as a function of treewidth. We also give the first polynomial-time algorithms on partial k-trees for certain problems, mainly Grundy Number, not known to have a finite state description even if restricted to graphs of bounded treewidth.

"Vertex Partitioning Problems: Characterization, Complexity and Algorithms on Partial k-trees" a dissertation prepared by Jan Arne Telle in partial fulfillment of the requirements for the Doctor of Philosophy degree in the Department of Computer and Information Science. This dissertation has been approved and accepted by:

Chair of the Examining Committee

Date

Commitee in charge: Dr. Andrzej Proskurowski, Chair
Dr. Eugene Luks
Dr. Arthur Farley
Dr. William Kantor

Accepted by:

Vice Provost and Dean of the Graduate School

Acknowledgements

Thanks Andrzej. My advisor Andrzej Proskurowski has provided excellent academic guidance, rock-solid support through rough waters, lasting friendship, colorful back-country trips and a steady supply of freshly brewed coffee. I have learned immeasurably much from him.

Thanks to the rest of my thesis committee: to Bill Kantor, to Gene Luks for always giving quality feedback on my theory seminar presentations, to Art Farley for showing how elegantly it can be done. Thanks to Ginnie Lo and Sanjay Rajopadhye for the years of monetary support, good collaboration and most of all good fun in the Oregami group. Thanks to Chris Wilson for the enduring teaching, above all at the Friday 5 o'clock class, one day I'll be back for a Hammerhead. Thanks to Jan Kratochvíl for teaching me about covering problems during the joint work leading to Chapter IV of this thesis, for also otherwise sharing his artistry and for leading me across the river. Thanks to Steve Hedetniemi for encouragement and for suggesting the Grundy Number problem. Thanks to Bob Beals for useful comments on this thesis.

For financial support of this research I thank the Norwegian Research Council for Science and Humanities (Norges Teknisk-Naturvitenskapelige Forskningsråd) and the National Science Foundation.

Thanks to the CIS department for admitting such a good bunch of students from all around the world. First of all, my officemate Sastry, I am very fortunate to have met you, thanks for everything. To Aseem, for showing me how to achieve balance. To Ferenc, for all the Hungarian jokes. To the other Indians, Samik, Renga, Sreeram, Joydip and Chandra. To Takunari from Japan, Judit from Hungary, Xiaoxiong and Guoqing from China. To Christof from Germany, to Susan, Bart, Brad and Kathy from the United States. To Juan from Mexico, to Lars Thomas and Roy from Norway, to Marek, and all the other honest philosophers with whom I have shared time, thanks.

Thanks Kari for the daily support and for the necessary counterbalance to the world of logic.

Contents

Chapter 1

Introduction

Many areas of computer science and many computer applications deal with systems best modeled as graphs, with vertices denoting entities and edges denoting relations between entities. The study of algorithmic solutions to graph problems is therefore of practical importance. While designing an algorithm for the maximum matching problem in 1965, Edmonds [30] defined the widely accepted notion of a *good* algorithm as one whose running time on any input is bounded by a polynomial function of the input size. A problem phrased as a yes/no question belongs to the class $\mathcal{P}$ if there is a good algorithm for solving it. A problem belongs to the class $\mathcal{NP}$ if any "yes" answer has a short proof that can be verified by a good algorithm. The $\mathcal{NP}$-complete problems are the hardest problems in $\mathcal{NP}$, as formulated by Cook in 1971 [25]. Perhaps the foremost open question in the theory of algorithms is whether $\mathcal{P} = \mathcal{NP}$; equivalently, whether all or none of the $\mathcal{NP}$-complete problems have good algorithms. The current belief is that $\mathcal{P} \neq \mathcal{NP}$. Unfortunately, many useful problems defined on graphs are $\mathcal{NP}$-complete. The results in this thesis can be viewed as addressing this situation in two ways: by discriminating graph problems with provably good algorithms from $\mathcal{NP}$-complete problems, and by designing good algorithms on restricted classes of graphs for problems which are generally $\mathcal{NP}$-complete. One of our contributions is a characterization of *vertex partitioning* problems which include, *e.g.*, coloring and domination problems. This characterization provides a basis for a taxonomy of vertex partitioning problems which we employ for, among other things, the study of their computational complexity in a unified framework. A second contribution is a template for the design of good algorithms on partial k-trees, equivalently graphs of bounded treewidth, which accounts for dependency on the treewidth k in both design and time complexity. These results are linked by partial k-tree algorithms for solving vertex partitioning problems, providing the first polynomial-time algorithms on partial k-trees for certain problems and the first careful investigation of time complexity as a function of the treewidth.

First, an overview of the presentation. The remainder of this introduction gives, after some basic definitions, the background for vertex partitioning problems and partial k-tree algorithms. Chapter II contains a general characterization of vertex partitioning problems, and also refined characterizations for vertex subset problems. These characterizations set the stage for results of subsequent chapters. We introduce several new classes of graph problems as generalizations of some classical problems admitting the characterization. In Chapter III, we concentrate on the complexity of vertex subset optimization problems, giving both efficient algorithms and $\mathcal{NP}$-completeness results for several infinite classes of problems. Chapter IV studies the complexity of the H-cover problem, which has a natural definition using our characterization. We develop a methodology that is useful in analyzing the complexity of H-covering problems, for any fixed graph H, and settle their complexity for any simple graph H on at most six vertices. Chapter V gives a methodology for the design of practical algorithms on partial k-trees based on a binary parse tree of the input graph. In Chapter VI we use this methodology to give partial k-tree algorithms first for vertex subset problems and then for the more general case of vertex partitioning problems. We conclude in Chapter VII by sketching some ideas for future research.

1.1 Definitions

We give some basic definitions relating to graphs and algorithms that we will use throughout the thesis. Notions exclusive to a particular chapter may not be defined here, *e.g.*, partial k-tree definitions can be found in the opening of Chapter V.

Let $\mathbb{N} = \{\nvdash, \nVdash, \nvDash, ...\}$ be the non-negative integers, and let $\mathbb{P} = \{\nVdash, \nvDash, \nVdash, ...\}$ be the positive integers. For sets X and Y, let $|X|$ be the cardinality of X, let $X \setminus Y = \{x \in X : x \notin Y\}$ and let $\binom{X}{k} = \{W : W \subseteq X \wedge |W| = k\}$ be the set of all k-element subsets of X. A *q-partition* $X_1, X_2, ..., X_q$ of the set X into q *classes* satisfies $X = \bigcup_{i \in \{1,...,q\}} X_i$ and $\emptyset = X_i \cap X_j, 1 \leq i \neq j \leq q$.

A *graph* $G = (V(G), E(G))$ is the pair of sets of *vertices* $V(G)$ and of *edges* $E(G)$, where $E(G) \subseteq \binom{V(G)}{2}$. Most of our results can be easily extended to directed graphs and to graphs containing loops and multiple edges, but they will not be considered here.

Two vertices $u, v \in V(G)$ are *adjacent* or *neighbors* if $uv \in E(G)$. For a vertex $v \in V(G)$, let $N_G(v) = \{u : uv \in E(G)\}$ be the set of neighbors of v and $deg_G(v) = |N_G(v)|$ its *degree*. We call $N_G(v) \cup v$ the *closed* neighborhood of the vertex v. A path of length k between vertices u and v is a sequence of distinct vertices $u = u_0, u_1, ..., u_k = v$ such that $u_{i-1}u_i \in E(G)$ for $1 \leq i \leq k$. The sequence of vertices forms a *cycle* of length $k+1$ if also $u_k u_0 \in E(G)$.

In a *connected* graph there is a path between any two vertices. A graph H is a *subgraph* of a graph G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$, it is a *spanning* subgraph if, in addition, $V(H) = V(G)$. A *component* in a graph is a maximal connected subgraph. The *distance*(u, v) between vertices u and v in the same component is the length of a shortest path between them.

A *tree* T is a connected graph without any cycles, we call its vertices *nodes*. Its *root* $r \in V(T)$ is a distinguished node by which for any $v \in V(T)$ we define *children*$(v) =$

$\{u : uv \in E(T) \wedge distance(u, r) = distance(v, r) + 1\}$. The complementary notion $parent(u)$ for $u \neq r$ is defined to be the unique node v for which $u \in children(v)$.

For $S \subseteq V(G)$ let $G[S] = (S, \{uv : u, v \in S \wedge uv \in E(G)\})$ denote the subgraph *induced* in G by S. For $S \subseteq V(G)$ let $G \setminus S = G[V(G) \setminus S]$, and for $F \subseteq E(G)$ let $G \setminus F = (V(G), \{uv \in E(G) : uv \notin F\})$. A *separator* of a graph G is a subset of vertices $S \subseteq V(G)$ such that $G \setminus S$ has more components than G.

Two graphs G and H are *isomorphic* if there is a bijection $f : V(G) \to V(H)$ such that $uv \in E(G) \Leftrightarrow f(u)f(v) \in E(H)$. The *automorphism group* of a graph G is the group $Aut(G)$ of permutations of $V(G)$ preserving adjacencies.

If $\forall v \in V(G) : deg_G(v) = k$ then G is k-regular. A $(|V(G)| - 1)$-regular graph G is a *complete* graph $K_{|V(G)|}$, also called a $|V(G)|$-*clique*. The 0-regular graph is called a *discrete* graph. A 1-regular graph is called a *perfect matching* and a 2-regular connected graph G a $|V(G)|$-cycle $C_{|V(G)|}$. A graph G is *bipartite* if $V(G)$ has a 2-partition V_1, V_2 with $E(G) \subseteq \{uv \in E(G) : u \in V_1 \wedge v \in V_2\}$. The *complement* graph of G is $\overline{G} = (V(G), \{uv : uv \notin E(G)\}$.

We give some definitions related to time complexity of algorithms. A *polytime* algorithm is one for which the number of steps executed on any input is bounded by a polynomial function of the input size. A *decision* problem is phrased as a yes/no question. Any optimization problem discussed in this thesis has a decision version, *e.g.*, for the optimization problem "given a graph G as input find the maximum length of any cycle in G", we have the decision version "given G and an integer k decide if G has a cycle of length at least k". It is not hard to show that these problems are polytime equivalent, in the sense that the optimization version has a polytime algorithm if and only if the decision version has one. For this reason, we may be imprecise and not distinguish carefully between an optimization problem and its decision version.

A decision problem belongs to the class $\mathcal{P}$ if it has a polytime algorithm and it belongs to the class $\mathcal{NP}$ if any "yes" answer has a short proof that can be verified by a polytime algorithm. The decision version of any problem addressed in this thesis belongs to $\mathcal{NP}$, *e.g.*, a short proof for the above example would be a sequence of vertices forming a cycle of length at least k. A polytime *reduction* from a problem A to a problem B is a polytime algorithm which takes an instance of A and outputs an instance of B such that their yes/no questions have identical answers. A problem B in $\mathcal{NP}$ is $\mathcal{NP}$-*complete* if for all problems A in $\mathcal{NP}$ there is a polytime reduction from A to B. Several $\mathcal{NP}$-complete problems are known. The $\mathcal{NP}$-completeness of a new $\mathcal{NP}$ problem C is demonstrated by a polytime *reduction* from a known $\mathcal{NP}$-complete problem B. The optimization version of an $\mathcal{NP}$-complete problem is in the class of $\mathcal{NP}$-*hard* problems.

1.2 Background on Vertex Partitioning Problems

A q-coloring of a graph is an assignment of one of q colors to each vertex of a graph so that no two adjacent vertices receive the same color. Coloring problems on graphs have been studied since the mid-1800s, starting with the famous Four-Color Conjecture that any planar graph could be 4-colored, resolved with the aid of a computer in 1976 [5]. The q-coloring problem asks whether an input graph has a q-coloring and is $\mathcal{NP}$-complete for any q greater than two. An important application is the compiler optimization problem of register allocation, modeled by representing variables as vertices and connecting vertices by an edge if the live program ranges of the corresponding variables overlap. A q-coloring of the resulting graph corresponds to an allocation of variables to q registers with no usage conflict. We will view a q-coloring as a partition $V_1, V_2, ..., V_q$ of the vertex set with the constraint that any vertex in V_i have no neighbors in V_i, for $1 \leq i \leq q$. Our characterization of vertex partitioning problems in Chapter II generalizes this constraint to allow for any specified number of neighbors the vertices in V_i can have in V_j, for $1 \leq i, j \leq q$. We show that many well-known problems admit such a characterization, and that we can define several new interesting graph parameters within this framework.

If we restrict attention to 2-partitions $(S, V(G) \setminus S)$ of vertices of a graph G and constrain only the number of neighbors in S we get a class of vertex subset problems which includes variants of domination and independence. Covering a chessboard by various pieces constitutes a precursor to the general theory of domination in graphs, with our compatriot Øystein Ore [56] being one of the pioneers in the field. A variety of special types of domination have been considered since, with applications to facility location and communication network problems. The current bibliography of papers related to the general topic of domination in graphs, by Hedetniemi and Laskar [40], has about 750 entries. A paper in the field of algorithmic theory of domination in graphs typically introduces a new domination-type parameter, contrasts it with related domination parameters and gives computational complexity results; all in a fairly ad-hoc manner. Upon the introduction of a slight variation of the parameter this work would then usually be repeated. In contrast, the characterization we propose in Chapter II facilitates the common algorithmic treatment of all these parameters and allows for their uniform complexity classification, the subject of Chapters III and VI. These parameters oftentimes arise from various fields, traditionally seen as separate, with the confusing effect that naming conventions and definitions are not standardized. The characterization suggested in Chapter II remedies this by explicitly focusing attention on the definitional properties of the parameters and on their relationships.

The vertex partitioning view can also be taken of covering problems on graphs. Let H be a graph with vertices $\{v_1, v_2, ..., v_q\}$. The H-cover problem takes a graph G as input and asks for a partition of the vertices of G into classes $V_1, ..., V_q$ such that if v_i is adjacent to v_j in H then any vertex in V_i has exactly one neighbor in V_j; otherwise there are no adjacencies between vertices in V_i and V_j. We trace H-coverings to Biggs' construction of highly symmetric graphs in [15], and to Angluin's discussion of "local knowledge" in distributed computing environment in [3]. More recently, Abello *et al.* [1] raised the question of computational complexity of H-cover problems, noting that there are both polynomial-time solvable and $\mathcal{NP}$-complete versions of this problem for different graphs H. A related question of complexity of H-coloring (also parametrized by a fixed graph H and definable in our characterization) has been resolved by Hell and Nešetřil [42] who completely classified graphs for which a polytime algorithm is known and those for which it is $\mathcal{NP}$-complete. In Chapter IV, we develop a methodology that is useful in analyzing the complexity of H-covering problems, and settle their complexity for any simple graph H on at most six vertices.

1.3 Background on Partial k-Tree Algorithms

Since the early days of graph algorithms it has been well known that most parameters are easily computed on trees. A combination of divide-and-conquer and dynamic programming techniques can contribute to finding an overall solution by recursively combining solutions to subproblems on subtrees. In 1982, Takamizawa, Nishizeki and Saito [62] extended these techniques to deal with many problems on the class of series-parallel graphs. The quest was on for the most general class of graphs sharing these algorithmic properties (see [58] for an overview.) Two independent lines of research led to the exact same answer, the partial k-trees (Arnborg and Proskurowski [9]) or equivalently graphs of treewidth bounded by k (Robertson and Seymour [59].) This class is a very promising generalization of trees and encompasses most other suggested classes.

A graph G is a k-tree if it is a complete graph on k vertices or if it has a vertex $v \in V(G)$ whose neighbors induce a clique of size k and $G \setminus \{v\}$ is again a k-tree. Partial k-trees are subgraphs of k-trees, and we note that any graph on n vertices is a partial k-tree for some value of k (the maximum value $k = n-1$ achieved by complete graphs.) Many natural classes of graphs have bounded treewidth [53], *e.g.*, trees are exactly the 1-trees and series-parallel graphs are partial 2-trees. Many optimization problems, while inherently difficult ($\mathcal{NP}$-complete) for general graphs are solvable in linear time on partial k-trees, for fixed values of k [11]. These solution algorithms have two main steps, first finding a parse tree (an embedding in a k-tree or a tree-decomposition of width k [59]) of the input graph, and then computing the solution by a bottom-up traversal of the parse tree. For the first step, Bodlaender [17] has given a linear algorithm deciding if a graph is a partial k-tree and if so finding a tree-decomposition of width k, for fixed k. Unfortunately, the complexity of this algorithm as a function of the treewidth does not make it practical for larger values of k. For $k \leq 4$, however, practical algorithms based on graph rewriting do exist for the first step [10, 54, 60].

There are many approaches for the design of the second step of partial k-tree algorithms with time complexity polynomial, or even linear, in the number of vertices [58, 7]. The strongest result in this direction by Courcelle and Mosbah [28] and Arnborg, Lagergren and Seese [8] states that any graph problem describable in a certain logic language, mainly EMSOL, has a polynomial-time algorithm on partial k-trees. As a rule, proponents of these approaches have tried to encompass as wide a class of problems as possible, often at the expense of increased complexity in k and also at the expense of simplicity of the resulting algorithms. Results giving explicit practical algorithms in this setting are usually confined to a few selected problems on either partial 1-trees or partial 2-trees [62, 36, 66]. In Chapter V, we try to cover the middle ground between these extremes and investigate both the practical design of algorithms for the second step and also their complexity, for varying k. The treewidth k is fixed for a given algorithm, but we analyze the complexity for growing values of this parameter. In the paradigm we suggest, the algorithm follows a binary parse tree of the input graph. This parse tree is based on very simple graph operations, facilitating the derivation of practical algorithms. We conclude our presentation in Chapter VI by applying this paradigm to vertex partitioning problems. These algorithms accept as input a graph G on n vertices and a width k tree-decomposition of G. We perform the first careful investigation of time complexity as a function of the treewidth for a general class of problems. For instance, the vertex subset optimization problems are solved in $T(n,k) = O(n2^{ck})$ time for small constants c. Since these problems are $\mathcal{NP}$-complete in general and a tree-decomposition of width $n-1$ is trivial for any graph, we cannot get polynomial dependence on both n and k, unless $\mathcal{P} = \mathcal{NP}$. Our results also include the first polynomial-time algorithms on partial k-trees for some problems that have not been found to be expressible in EMSOL [50], and not known to have finite-state descriptions, mainly the Grundy Number problem. This follows from (i) the description of the Grundy Number problem as a vertex partitioning problem, (ii) a new logarithmic bound on the Grundy Number of a partial k-tree, and (iii) our investigation of time complexity of vertex partitioning problems on partial k-trees.

Chapter 2

Characterization of Vertex Partitioning Problems

We define vertex partitioning problems and show that many well-known problems, such as coloring and covering, admit a characterization as such problems. Many of these problems, including variations of domination, independence and packing, are defined over 2-partitions of vertices. For these vertex subset optimization problems we give a separate characterization, mainly for notational purposes. We then extend this vertex subset characterization to encompass irredundance-type problems, defined over maximal and minimal vertex subsets. These characterizations provide a basis for a taxonomy of vertex partitioning problems, facilitating their common algorithmic treatment and allowing for their uniform complexity classification, the subject of subsequent chapters. In this chapter we show applicability of the characterization by introducing some non-trivial new graph problems as variations of classic problems. We conclude the chapter with an application of the characterization to a graph-theoretic question.

2.1 General Vertex Partitioning Problems

A q-coloring of a graph is a partition $V_1, V_2, ..., V_q$ of its vertices where any vertex in V_i has no neighbors in V_i, for $1 \leq i \leq q$. In the following we generalize this constraint to allow for any specified number of neighbors the vertices in V_i can have in V_j, for $1 \leq i, j \leq q$.

> **Definition 2.1** A *degree constraint* matrix D_q is a q by q matrix with entries being subsets of $\mathbb{N} = \{0, 1, 2, ...\}$. A *$D_q$-partition* in a graph G is a q-partition $V_1, V_2, ..., V_q$ of $V(G)$ such that for $1 \leq i, j \leq q$ we have $\forall v \in V_i : |N_G(v) \cap V_j| \in D_q[i, j]$.

For technical reasons, we will allow the possibility of some $V_i = \emptyset$ in a D_q-partition $V_1, ..., V_q$. We limit attention to non-empty graphs, $|V(G)| \geq 1$. For a simple example,

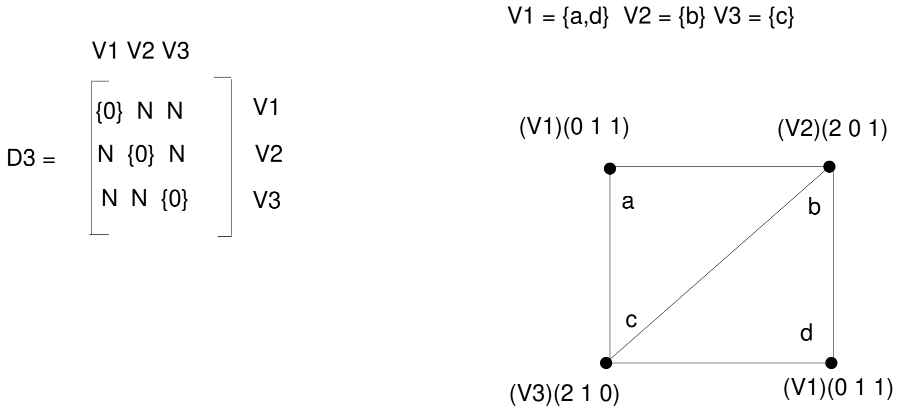


Figure 2.1: The degree constraint matrix D_3 for deciding if there exists a 3-coloring (N=$\{0, 1, 2, ...\}$). Also, a given partition on a graph, with vertices of the graph labeled by the class they belong to (Vi) and a 3-vector (a b c) giving the number of neighbors it has in classes V1, V2 and V3, respectively. Note that each vertex satisfies the constraint imposed by D_3, so this partition is a 3-coloring of the graph.

using a degree matrix D_1 with entry $\{k\}, k \in \mathbb{N}$, a graph G will have a D_1-partition iff it is k-regular. We define problems over D_q-partitions as follows:

Definition 2.2 For fixed degree constraint matrices $D_1, D_2, ...$ and a given graph G:

For fixed q, the $\exists D_q$-problem decides if G has a D_q-partition.

The $minD_q$-problem asks for the minimum q such that G has a D_q-partition.

The $maxD_q$-problem asks for the maximum q such that G has a D_q-partition.

We next show several well-known graph problems defined in this framework. [1]

- The q-COLORING problem [GT4] is the $\exists D_q$-problem over the degree constraint matrix with diagonal entries $\{0\}$ and off-diagonal entries $\mathbb{N}$. See Figure 2.1 for an example.
- The CHROMATIC NUMBER problem [GT4] is the $minD_q$-problem over matrices with diagonal entries $\{0\}$ and off-diagonal entries $\mathbb{N}$.
- The DOMATIC NUMBER problem [GT3] is the $maxD_q$-problem over matrices with diagonal entries $\mathbb{N}$ and off-diagonal entries $\mathbb{P}$. Note that the constraints imposed by D_q in this case will enforce all partition classes to be non-empty.

[1] [GTx] as a citation refers to the Graph Theory problem number x in Garey and Johnson [34]

- The PARTITION INTO PERFECT MATCHINGS problem [GT16] is the $minD_q$-problem over matrices with diagonal entries $\{1\}$ and off-diagonal entries $\mathbb{N}$.
- The GRAPH GRUNDY NUMBER problem [GT56, undirected version] is the $maxD_q$-problem over matrices with diagonal entries $\{0\}$, above-diagonal entries $\mathbb{N}$ and below-diagonal entries $\mathbb{P}$. For this definition we must explicitly add the requirement that a D_q-partition $V_1, ..., V_q$ have only non-empty partition classes.
- The H-COVER problem [3] is the $\exists D_{|V(H)|}$-problem with $D_{|V(H)|}$ the adjacency matrix of H (with singleton entries $\{0\}$ and $\{1\}$).
- The H-COLOR problem [42] is the $\exists D_{|V(H)|}$-problem with $D_{|V(H)|}$ the matrix obtained from the adjacency matrix of H by replacing 1-entries with $\mathbb{N}$ and 0-entries with $\{0\}$.
- The VERTEX SUBSET problems defined in the next section, see Table 2.2, have degree constraint matrices D_2 of the form

$$\begin{pmatrix} \sigma & \mathbb{N} \\ \rho & \mathbb{N} \end{pmatrix}$$

Most of these definitions follow immediately from the standard definitions of the problems. The GRAPH GRUNDY NUMBER problem is traditionally defined as a coloring problem where vertices are colored using non-negative integers, in such a way that a vertex with color i is forced to have neighbors with colors 1 through $i-1$. The problem asks for the highest color we can use while observing this constraint. In our characterization, the partition class V_i is the set of vertices with color i, with the constraint that a vertex in V_i have no neighbors in V_i and at least one neighbor in each of the sets $V_{i-1}, V_{i-2}, ..., V_1$. Since we are looking for the highest number of partition classes possible, we require that all classes be non-empty. This constraint is not enforced by the degree constraint matrix itself, as it is in the case of DOMATIC NUMBER, thus it must be added explicitly to the definition of the problem. We return to this definition of GRAPH GRUNDY NUMBER in section 4 of this chapter.

The H-COLOR problem asks for the existence of a labeling $f : V(G) \rightarrow V(H)$ such that $uv \in E(G) \Rightarrow f(u)f(v) \in E(H)$. In our characterization, we fix an ordering $V(H) = \{v_1, v_2, ..., v_{|V(H)|}\}$ with the partition class V_i the set of vertices labeled v_i. Thus the constraint of the partition is that for all pairs (i, j) with $i \neq j$ and $v_j \notin N_H(v_i)$, no vertex in V_i is adjacent to a vertex in V_j. To frame this as an $\exists D_q$ problem, like we did above, requires that we do allow a partition $V_1, ..., V_q$ to have some empty partition classes. The H-COVER problem is examined in detail in chapter 4.

We discuss some problems definable by extensions of the given vertex partition characterization. The first extension involves optimizations over the cardinality of certain partition classes, the main optimization concern of the vertex subset problems dealt with in the next section. The DISTANCE $\leq q$ DOMINATION problem [51] asks for the smallest vertex subset S with the property that any vertex $x \notin S$ have a neighbor in S at distance $\leq q$. This can be defined as an $\exists D_{q+1}$ problem over the degree constraint matrix D_{q+1} with diagonal and above-diagonal entries $\mathbb{N}$, entries directly below the diagonal $\mathbb{P}$ and remaining entries $\{0\}$. For a D_{q+1} partition $V_1, ..., V_{q+1}$, the vertex subset V_1 will have the required domination property, with vertices in class V_j at distance $j-1$ from some vertex in V_1. The problem is thus defined by minimizing $|V_1|$ over all D_{q+1}-partitions $V_1, V_2, ..., V_{q+1}$. Note that this definition allows for classes $V_i, V_{i+1}, ..., V_{q+1}, i \geq 2$ to all be empty.

For the second extension, we allow entries of the matrix to be simple arithmetic expressions involving the cardinality of a partition class. This allows the definition of *e.g.* PARTITION INTO CLIQUES [GT15]. For example, a graph is a SPLIT graph [35] if its vertices can be partitioned into a clique and an independent set or equivalently if it has a D_2-partition with $D_2[1,1] = |V_1| - 1, D_2[2,2] = \{0\}, D_2[1,2] = D_2[2,1] = \mathbb{N}$.

The problems BALANCED COMPLETE BIPARTITE SUBGRAPH [GT24] and MAXIMUM CLIQUE [GT19] can be defined if we allow both extensions discussed above.

To express the PARTITION INTO TRIANGLES problem [GT11], we must enforce non-empty partition classes, and allow the size q of the matrix D_q to be a function of the size of the input graph, in this case $q = |V(G)|/3$, with the matrix containing $\{2\}$ on the diagonal and $\mathbb{N}$ off the diagonal.

In the next section, we consider problems defined over D_2-partitions with constraints only on the number of neighbors in V_1, *i.e.*, $D_2[1,2] = D_2[2,2] = \mathbb{N}$.

2.2 Vertex Subset Problems

If every vertex in a selected subset S of vertices of a graph has zero selected neighbors then S is an independent set, and similarly if every vertex not in S has at least one selected neighbor then S is a dominating set. This suggests a common characterization of independent sets and dominating sets based on the constraints imposed on the number of selected neighbors the vertices in S, and vertices not in S, can have.

Let the symbols σ and ρ indicate membership in S and membership in $V(G) \setminus S = \{v \in V(G) : v \notin S\}$, respectively.

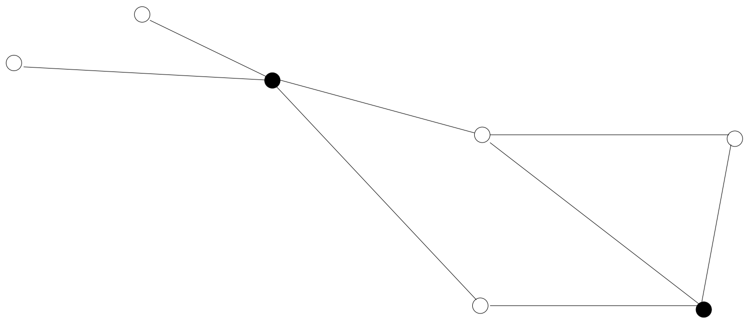


Figure 2.2: Dark vertices form the vertex subset S; vertices labelled by $state_S$

Definition 2.3 Given a graph G and a set $S \subseteq V(G)$ of *selected* vertices

- The *state* of a vertex $v \in V(G)$ is

$$state_S(v) \stackrel{df}{=} \begin{cases} \rho_i & \text{if } v \notin S \text{ and } |N_G(v) \cap S| = i \\ \sigma_i & \text{if } v \in S \text{ and } |N_G(v) \cap S| = i \end{cases}$$

- Define syntactic abbreviations

  $\rho_{\leq i} \equiv \rho_0, \rho_1, ..., \rho_i$ $\quad \sigma_{\leq i} \equiv \sigma_0, \sigma_1, ..., \sigma_i$ $\quad \rho_{\geq i} \equiv \rho_i, \rho_{i+1}, ...$ $\quad \sigma_{\geq i} \equiv \sigma_i, \sigma_{i+1}, ...$

Each of the latter two abbreviations represents an infinite set of states. Mnemonically, σ represents a vertex selected for S and ρ a vertex rejected from S, with the subscript indicating the number of neighbors the vertex has in S. See Figure 2.2 for an example. A variety of vertex subset properties can be defined by allowing only a specific set L as *legal* states of vertices. For instance, S is a dominating set if state ρ_0 is not allowed for any vertex, giving the legal states $L = \{\rho_{\geq 1}, \sigma_{\geq 0}\}$. Table 2.1 relates some of the established terminology to our formalism. For example, the subset shown in Figure 2.2 is an independent dominating set. Optimization problems over these sets often maximize or minimize the size of the set of vertices with states in a given $M \subseteq L$. For instance, in the minimum dominating set problem, $M = \{\sigma_{\geq 0}\}$.

Term	Expressed in our formalism
Dominating	ρ_0 not a legal state
Independent	σ_0 the only legal σ-state
Perfect Dominating	ρ_1 the only legal ρ-state
Nearly Perfect	ρ_0 and ρ_1 the only legal ρ-states
Total P	effect of property P on ρ-states is extended to σ-states
Induced	$\rho_{\geq 0}$ legal state (no non-legal ρ-states)

Table 2.1: The established terminology and our formalism

Definition 2.4 Given sets M and L of vertex states and a graph G:

- $S \subseteq V(G)$ is an $[L]$-set if $\forall v \in V(G) : state_S(v) \in L$;
- $\exists[L]$ is the problem asking whether there exists any $[L]$-set $S \subseteq V(G)$;
- $minM[L]$ (or $maxM[L]$) is the problem of minimizing (or maximizing) $|\{v : state_S(v) \in M\}|$ over all $[L]$-sets $S \subseteq V(G)$;
- $min[L]$ (or $max[L]$) is shorthand for $minM[L]$ (or $maxM[L]$) when M consists of all σ-states in L, in effect optimizing the size of the selected set of vertices.

Thus, a dominating set is a $[\rho_{\geq 1}, \sigma_{\geq 0}]$-set, with the square brackets implying the set notation. Table 2.2 shows some of the classical vertex subset properties [34, 23, 32, 12, 24, 33, 22, 44].

Our notation	Standard terminology
$[\rho_{\geq 0}, \sigma_0]$-set	Independent set
$[\rho_{\geq 1}, \sigma_{\geq 0}]$-set	Dominating set
$[\rho_{\leq 1}, \sigma_0]$-set	Strong Stable set or 2-Packing
$[\rho_1, \sigma_0]$-set	Efficient Dominating set or Perfect Code
$[\rho_{\geq 1}, \sigma_0]$-set	Independent Dominating set
$[\rho_1, \sigma_{\geq 0}]$-set	Perfect Dominating set
$[\rho_{\geq 1}, \sigma_{\geq 1}]$-set	Total Dominating set
$[\rho_1, \sigma_1]$-set	Total Perfect Dominating set
$[\rho_{\leq 1}, \sigma_{\geq 0}]$-set	Nearly Perfect set
$[\rho_{\leq 1}, \sigma_{\leq 1}]$-set	Total Nearly Perfect set
$[\rho_1, \sigma_{\leq 1}]$-set	Weakly Perfect Dominating set
$[\rho_{\geq 0}, \sigma_{\leq p}]$-set	Induced Bounded-Degree subgraph
$[\rho_{\geq p}, \sigma_{\geq 0}]$-set	p-Dominating set
$[\rho_{\geq 0}, \sigma_p]$-set	Induced p-Regular subgraph

Table 2.2: Some vertex subset properties.

Table 2.2 can be used as a quick reference guide to the exact definitions of the various properties represented and their derived problems. Naming conventions are not standardized. As an example, Biggs [15] and later Kratochvíl [46] consider Perfect Codes in graphs (as a generalization of error-correcting codes), Bange et al. [12] study Efficient Dominating Sets in graphs (a variant of domination), and Fellows et al. [32] investigate what they call Perfect Dominating Sets. In fact, they are all studying the exact same property, namely $[\rho_1, \sigma_0]$-sets.

In the next chapter we study the computational complexity of the problems defined over these and other vertex subset properties, see Table 3.1. Properties traditionally defined using closed neighborhoods are easily captured by the characterization. The vertex weighted versions of these parameters will optimize the sum of

Our notation	Standard terminology
$\exists[\rho_1, \sigma_0]$	Perfect Code Problem
$min[\rho_{\geq 1}, \sigma_{\geq 0}]$	Minimum Dominating Set Problem
$max[\rho_{\geq 0}, \sigma_0]$	Maximum Independent Set Problem
$min\{\rho_{\geq 0}\}[\rho_{\geq 0}, \sigma_0]$	Minimum Vertex Cover Problem
$max\{\rho_1\}[\rho_{\geq 0}, \sigma_{\geq 0}]$	Efficiency Problem

Table 2.3: Examples of graph problems.

the weights of vertices with state in M, with the cardinality version corresponding to unit weights. For directed graphs we consider $N_G(v)$ as $\{u : \langle u, v\rangle \in \text{Arcs}(G)\}$ to obtain directed versions of these domination-like properties and parameters. Table 2.3 shows examples of graph problems [13, 66] expressed using our characterization. Note that complementary problems, *e.g.* Maximum Independent Set and Minimum Vertex Cover, are both expressible.

2.3 Maximal and Minimal Vertex Subset Problems

We give a refinement of the vertex subset characterization of the last section, useful for describing maximal and minimal vertex subsets with a given property.

Definition 2.5 Given a set L of vertex states and a graph G

- $S \subseteq V(G)$ is a maximal (minimal) $[L]$-set if there is no vertex $v \notin S$ ($v \in S$) such that $S \cup \{v\}$ ($S \setminus \{v\}$) is an $[L]$-set.

Parameters related to irredundant sets in graphs are also expressible using the refinement. Irredundant sets require some vertices to have at least one neighbor with a given state. This motivates the definition of a *refined* vertex state as the juxtaposition, denoted by $\cdot$, of the state of the vertex with the state of one of its neighbors.

Definition 2.6 Given a graph G and a selected set of vertices $S \subseteq V(G)$:

- The set of *refined* vertex states of $v \in V(G)$ is $rstate_S(v) = \{state_S(v)\} \cup \{state_S(v) \cdot state_S(w) : w \in N(v)\}$;
- For a set R of refined states, S is an $[R]$-set if $\forall v \in V(G) : rstate_S(v) \cap R \neq \emptyset$;
- For sets R and M of vertex states $minM[R]$ ($maxM[R]$) is the parameter minimizing (maximizing) $|\{v : rstate_S(v) \cap M \neq \emptyset\}|$ over all $[R]$-sets S.

Our notation	Standard terminology
$[\rho_{\geq 0}, \sigma_0, \sigma_{\geq 1} \cdot \rho_1]$-set	Irredundant set (closed-closed)
$[\rho_{\geq 0}, \sigma_0, \sigma_{\geq 1} \cdot \rho_1, \sigma_{\geq 1} \cdot \sigma_1]$-set	closed-open Irredundant set
$[\rho_{\geq 0}, \sigma_{\geq 0} \cdot \rho_1, \sigma_{\geq 0} \cdot \sigma_1]$-set	open-open Irredundant set
$[\rho_{\geq 0}, \sigma_{\geq 0} \cdot \rho_1]$-set	open-closed Irredundant set
$[\rho_{\geq 0}, \sigma_{\leq k-1}, \sigma_{\geq k} \cdot \rho_k]$-set	k-Irredundant set
$[\rho_{\geq 1}, \sigma_0, \sigma_{\geq 1} \cdot \rho_1]$-set	Minimal Dominating set
$[\rho_{\leq 1} \cdot \rho_1, \sigma_{\geq 0}]$-set	Maximal Nearly Perfect set
$max\{\sigma_{\geq 0}\}[\rho_{\geq 0}, \sigma_0, \sigma_{\geq 1} \cdot \rho_1]$	Upper Irredundance parameter
$max\{\sigma_{\geq 0}\}[\rho_{\geq 1}, \sigma_0, \sigma_{\geq 1} \cdot \rho_1]$	Upper Dominating parameter

Table 2.4: Some vertex subset properties and graph parameters defined using refined states

Abbreviations like $\sigma_{\geq 1}{\cdot}\rho_1$ denote $\sigma_1{\cdot}\rho_1, \sigma_2{\cdot}\rho_1, \ldots$, in analogy with earlier definitions. For example, irredundant sets have legal refined states $R = \{\rho_{\geq 0}, \sigma_0, \sigma_{\geq 1}{\cdot}\rho_1\}$, meaning that for an R-set $S \subseteq V(G)$ we have $states_S(v) \in \{\sigma_1, \sigma_2, ...\} \Rightarrow \exists w \in N_G(v) : states_S(w) = \rho_1$ (a selected vertex having at least one selected neighbor must also have a private non-selected neighbor.)

Table 2.4 gives examples of vertex subset properties and graph parameters [24, 31, 33, 44] admitting a characterization using refined states. The discriminating term "closed-closed" for irredundant sets arises from the definition of an irredundant set S as one for which $\forall v \in S$ the union of the closed neighborhoods of vertices in $S \setminus \{v\}$ is strictly smaller than the union of the closed neighborhoods of vertices in S.

For a given set of (non-refined) vertex states L we now give a general procedure constructing sets of refined vertex states $Lmax$ and $Lmin$ such that the $[Lmax]$-sets are exactly the maximal $[L]$-sets and the $[Lmin]$-sets are exactly the minimal $[L]$-sets. Given L, we define the following vertex states:

$$Amax = \{\rho_i : \rho_i \in L \wedge \sigma_i \notin L\}$$
$$Amin = \{\sigma_i : \sigma_i \in L \wedge \rho_i \notin L\}$$
$$Bmax = \{\rho_i : \rho_i \in L \wedge \rho_{i+1} \notin L\} \cup \{\sigma_i : \sigma_i \in L \wedge \sigma_{i+1} \notin L\}$$
$$Bmin = \{\rho_i : \rho_i \in L \wedge \rho_{i-1} \notin L\} \cup \{\sigma_i : \sigma_i \in L \wedge \sigma_{i-1} \notin L\}$$

Let $L\rho$ and $L\sigma$ be the sets of ρ-states and σ-states in L, respectively, so that $L = L\rho \cup L\sigma$. We define states for maximal and minimal $[L]$-sets as follows:

$$Lmax \stackrel{df}{=} Amax \cup L\sigma \cup \{a \cdot b : a \in L\rho \setminus Amax \wedge b \in Bmax\}$$
$$Lmin \stackrel{df}{=} Amin \cup L\rho \cup \{a \cdot b : a \in L\sigma \setminus Amin \wedge b \in Bmin\}$$

Theorem 2.1 A vertex subset S is a maximal (respectively, minimal) $[L]$-set in G if and only if S is a $[Lmax]$-set (respectively, $[Lmin]$-set) in G.

Proof. We argue only for maximal sets as the proof for minimal sets is very similar. Let S be a maximal $[L]$-set in G. We show that $rstate_S(v) \cap Lmax$ is non-empty for any $v \in V(G)$. Since $state_S(v) \in rstate_S(v)$ it suffices to show $state_S(v) \in Lmax$. If $v \in S$ then the above clearly holds since $state_S(v) \in L\sigma \subseteq Lmax$. If $v \notin S$ then $S' = S \cup \{v\}$ is not an $[L]$-set so there exists at least one vertex u with $state_{S'}(u) \notin L$. Since $state_S(w) = state_{S'}(w)$ for any $w \notin N_G(v) \cup \{v\}$ we consider two cases. case (i) $state_{S'}(v) \notin L$. Let $state_S(v) = \rho_i$ so that $state_{S'}(v) = \sigma_i \notin L$. But then $state_S(v) \in Amax \subseteq Lmax$. case (ii) $state_{S'}(v) \in L$. We have $state_{S'}(u) \notin L$ for some $u \in N_G(v)$ and either $u \in S$ or $u \notin S$. We argue only for $u \in S$ as the reasoning for $u \notin S$ is very similar. Let $state_S(u) = \sigma_i$ so that $state_{S'}(u) = \sigma_{i+1} \notin L$. Note that $state_S(u) \in Bmax$ and $state_S(v) \in L\rho \setminus Amax$ so that among the refined states $rstate_S(v)$ of vertex v we have $state_S(v) \cdot state_S(u) \in Lmax$. We leave out the other direction of the proof as it is basically a reversal of the above arguments. □

As an example, Table 2.4 shows the resulting characterizations for minimal dominating sets and maximal nearly perfect sets. Note that maximal $[L]$-sets (similarly, minimal $[L]$-sets) are exactly the $[L]$-sets themselves if $Amax$ ($Amin$) contains every ρ-state (every σ-state) in L or if $Bmax = L$ ($Bmin = L$) and the graph G has no isolated vertices.

2.4 Some New Vertex Partitioning Problems

We define some new vertex partitioning problems as generalizations of the old problems encountered in earlier sections. Several new vertex subset problems are introduced in the next chapter. For instance, the new vertex subset problem $max\{\rho_1, \sigma_1\}[\rho_{\geq 0}, \sigma_{\geq 0}]$, which we call TOTAL EFFICIENCY is derived from the old EFFICIENCY [13] problem $max\{\rho_1\}[\rho_{\geq 0}, \sigma_{\geq 0}]$. This problem arises in communication networks, if we assume that a communication round has two time-disjoint phases, *send* and *receive*, and that a processor receives a message whenever it has a single sending neighbor. The maximum number of processing elements that can receive a message in one communication round is the Total Efficiency of the graph underlying the network topology.

The CHROMATIC number problem is the problem of minimizing the number of independent sets the vertices of a graph can be partitioned into. Similarly, DOMATIC number maximizes the number of dominating sets, PARTITION INTO PERFECT MATCHING minimizes the number of induced 1-regular subgraphs and q-COLORING asks about the existence of a partition into q independent sets. For each vertex subset property in Table 2.2 and also Table 2.4 we can similarly define a partition maximization, a partition minimization and q-partition existence problems. We call the resulting problems $[\rho, \sigma]$-PARTITION problems. By a $[\rho, \sigma]$-property we will mean the property enforced by the degree constraint matrix

$$\begin{pmatrix} \sigma & \mathbb{N} \\ \rho & \mathbb{N} \end{pmatrix}$$

For a $[\rho, \sigma]$-property we define these partition problems simply by taking the degree constraint matrix D_q with diagonal entries σ and off-diagonal entries ρ.

For example, the $[\rho_1, \sigma_0]$-PARTITION problem asking for the existence of a q-partition turns out to be exactly the K_q-COVER problem, solvable in polynomial time for $q \leq 3$, but $\mathcal{NP}$-complete otherwise. It may be interesting to investigate the cutoff points at which the q-partition existence problems for various vertex subset properties become intractable.

Let us consider a $[\rho_1, \sigma_{\geq 0}]$-PARTITION problem. We define the PERFECT MATCHING CUT problem as the $\exists D_2$ problem with D_2 the degree constraint matrix

$$\begin{pmatrix} \mathbb{N} & 1 \\ 1 & \mathbb{N} \end{pmatrix}$$

In other words, does the graph have a subset of vertices $S \subseteq V(G)$ such that the spanning subgraph on edges crossing the cut $(S, V(G) \backslash S)$ is 1-regular? As an example, the binomial trees and also the hypercube graphs have a perfect matching cut, which follows immediately from their iterative definition. We have not found any earlier references to this problem.

A more general class of problems arises if we consider partitions into *different* vertex subset properties, *e.g.* a SPLIT graph is one which has a partition into an independent set and a clique. In general, take vertex subset properties $[1\rho, 1\sigma], [2\rho, 2\sigma], ..., [q\rho, q\sigma]$, and construct a degree constraint matrix D_q with column i having entry $i\sigma$ on the diagonal (position i) and $i\rho$ off the diagonal. The $\exists D_q$-problem asks if a graph G has a partition $V_1, V_2, ..., V_q$ of $V(G)$ where V_i is an $[i\rho, i\sigma]$-set in G. We call these NON-UNIFORM PARTITION problems.

A variation of these problems arises by asking if a graph G has a partition $V_1, V_2, ..., V_q$ where V_i is a $[\rho, \sigma]$-set in $G \setminus (\cup V_j, j < i)$. To define this we use the degree constraint matrix D_q with diagonal entries σ, above-diagonal entries $\mathbb{N}$ and below-diagonal entries ρ. We call the resulting problems $[\rho, \sigma]$-REMOVAL problems, since V_1 is a $[\rho, \sigma]$-set in $G_1 = G$, while V_2 is a $[\rho, \sigma]$-set in $G_2 = G_1 \setminus V_1$, and in general V_i is a $[\rho, \sigma]$-set in $G_i = G_{i-1} \setminus V_{i-1}$. Here we may have to add the requirement that all partition classes be non-empty. For example, the $[\rho_{\geq 1}, \sigma_0]$-REMOVAL problem ($[\rho_{\geq 1}, \sigma_0]$-sets are Independent Dominating sets) asking for the maximum q such that a graph has the appropriate D_q-partition, with non-empty classes, is exactly the GRAPH GRUNDY NUMBER problem.

For another example, we consider a $[\rho_{\geq 1}, \sigma_{\geq 0}]$-REMOVAL problem. Define matrices $D_1, D_2, ...$ with below-diagonal entries $\mathbb{P}$ and with diagonal and above-diagonal entries $\mathbb{N}$. We call the $maxD_q$-problem over these matrices the UPPER-DOMINATING-REMOVAL problem. This parameter is the maximum number of times we can repeatedly remove dominating sets from a graph, before the graph becomes empty.

2.5 A Non-Algorithmic Application

As an example of a non-algorithmic application of our characterization, we consider a generalization of perfect codes ($[\rho_1, \sigma_0]$-sets) and extend to this generalization a result that holds for perfect codes.

> **Lemma 2.1** For $p \in \mathbb{P}, q \in \mathbb{N}$ if both A and B are $[\rho_p, \sigma_q]$-sets of a graph G then $|A| = |B|$.

Proof: Let $X_I = A \cap B$, $X_A = A \setminus B$ and $X_B = B \setminus A$, so that X_A, X_I, X_B is a partition of $A \cup B$. By a counting argument, we will show that $|X_A| = |X_B|$. Consider the edge-disjoint subgraphs $F = (X_A \cup X_B, \{uv \in E(G) : u \in X_A \wedge v \in X_B\})$ and $H = (A \cup B, \{uv \in E(G) : (u \in X_I \wedge v \in X_B) \vee (u \in X_I \wedge v \in X_A)\})$. Note F contains the edges between X_A and X_B while H contains the edges with one endpoint in X_I and the other endpoint in X_A or X_B. Since A, B are $[\rho_p, \sigma_q]$-sets, we have $\forall v \in X_A \cup X_B : deg_F(v) + deg_H(v) = p$. Since F is a bipartite graph we have $\sum_{v \in X_A} deg_F(v) = \sum_{v \in X_B} deg_F(v)$. Since A, B are $[\rho_p, \sigma_q]$-sets, both $G[A]$ and $G[B]$ are q-regular, so we have $\forall v \in X_I : |N(v) \cap X_A| = |N(v) \cap X_B|$, which gives $\sum_{v \in X_A} deg_H(v) = \sum_{v \in X_B} deg_H(v)$. But then

$$p|X_A| = \sum_{v \in X_A} deg_F(v) + \sum_{v \in X_A} deg_H(v) = \sum_{v \in X_B} deg_F(v) + \sum_{v \in X_B} deg_H(v) = p|X_B|$$

and since $p > 0$ we have $|X_A| = |X_B|$ which implies $|A| = |B|$. □

> **Theorem 2.2** For a set of vertex states L, the statement "For any graph G, all $[L]$-sets have the same size" is true if and only if (i) or (ii) holds
>
> (i) $L = \{\rho_p, \sigma_q\}$ for some $p \in \mathbb{P}, q \in \mathbb{N}$
>
> (ii) L has either no ρ-states or no σ-states

Proof: If L has no ρ-states then the only possible $[L]$-set is $S = V(G)$ and if L has no σ-states then the only possible $[L]$-set is $S = \emptyset$. The sufficiency of (i) and (ii) then follows from Lemma 2.1. For necessity we will consider sets L not of type (i) or (ii), and construct graphs with two $[L]$-sets of different sizes. First note that if $\rho_0 \in L$ then $S = \emptyset$ is an $[L]$-set and it is easy to construct a graph with some larger $[L]$-set. The remaining cases (when $\rho_0 \notin L$) are covered by two arguments, depending on whether there is more than one legal state for selected vertices, or more than one legal state for non-selected vertices. In both cases, we construct a graph G with appropriate subsets A and B (each set inducing a collection of complete graphs) of different sizes.

Case 1: Suppose $\{\rho_a, \rho_b, \sigma_c\} \subseteq L$ where $a < b$. For A and B disjoint, let $G = (A \cup B, E)$ where A induces a copies of K_{c+1} and B induces b copies of K_{c+1}, clearly both c-regular. The remaining edges form a perfect matching between each pair of

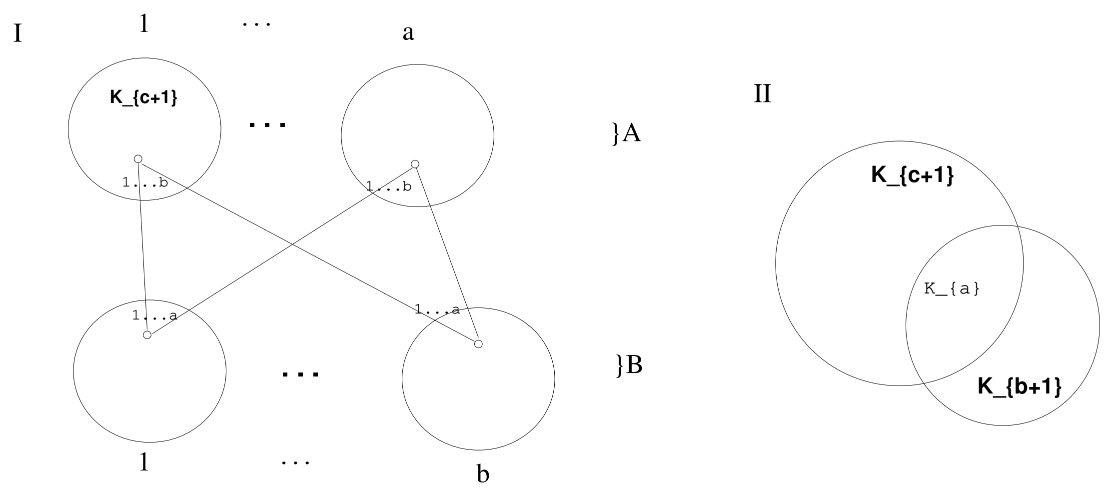


Figure 2.3: I) A graph having $[\rho_a, \rho_b, \sigma_c]$-sets A and B. II) A graph having two $[\rho_a, \sigma_b, \sigma_c]$-sets of size $b+1$ and $c+1$ $(a \le b+1)$

K_{c+1}'s, one from each of A and B. See Figure 2.3-I. Thus a vertex in A has b neighbors in B and a vertex in B has a neighbors in A. $|A| = a(c+1) < b(c+1) = |B|$ since $a < b$.

Case 2: Suppose $\{\rho_a, \sigma_b, \sigma_c\} \subseteq L$ where $b < c$. If $a \le b+1$ let $G = (A \cup B, E)$ such that A and B induce K_{b+1} and K_{c+1}, respectively, and $A \cap B$ induces K_a, these adjacencies accounting for all the edges. See Figure 2.3-II. If $a > b+1$ we use the graph depicted in Figure 2.4. As before, A induces the K_{c+1}'s and B induces the K_{b+1}'s (shaded in the figure). The remaining edges are between $A \setminus B$ and $B \setminus A$ and can be added in any way such that each vertex of $A \setminus B$ gets $a-b-1$ additional edges and each vertex of $B \setminus A$ gets a additional edges. Thus, the bipartite graph between $A \setminus B$ and $B \setminus A$ must have $(c+1-(b+1))(a-b-1)a(b+1) = (b+1)a(a-b-1)(c-b)$ edges, counting from $A \setminus B$ or $B \setminus A$ respectively, and since $a-b-1 < a$ we have $|A| > |B|$. $\square$

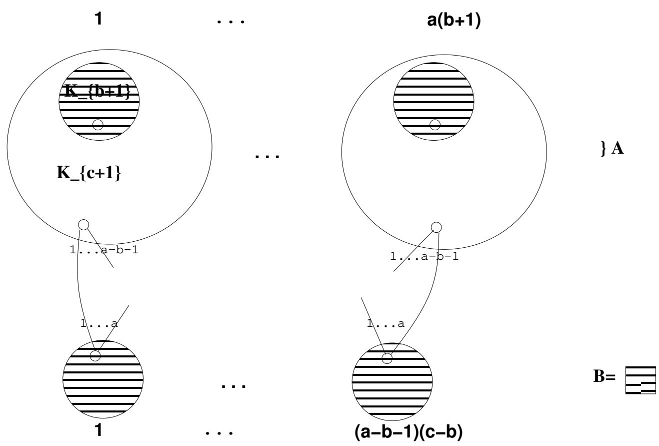


Figure 2.4: A graph having $[\rho_a, \sigma_b, \sigma_c]$-sets A and B $(a > b + 1)$

Chapter 3

Complexity of Vertex Subset Optimization Problems

We study the computational complexity of vertex subset optimization problems in a unified framework, using the characterization given in Chapter 2.2. In recent years, a variety of domination-type parameters in graphs have been introduced, and the number of papers devoted to this topic is steadily increasing [39, 40]. We give a table cataloging the computational complexity of computing some of these, and other, parameters. We also investigate the computational complexity of the general class of all problems admitting our characterization. For a given vertex subset property, we concentrate on the existence, maximization and minimization problems. The existence problem merely decides if a graph has any vertex subset with the property, while the maximum and minimum problems can be used to find the largest and smallest such vertex subset. Several infinite classes of $\mathcal{NP}$-complete and of polynomial time solvable problems are shown. We completely resolve the complexity of the existence version for those problems having a finite number of legal states, up to $\mathcal{P}$ vs. $\mathcal{NP}$. We also give $\mathcal{NP}$-completeness results for the existence version of problems with an infinite number of legal states, *e.g.*, deciding if a graph has a $[\rho_{\geq 1}, \sigma_1]$-set, which we call a Dominating Induced Matching. For some problems we show $\mathcal{NP}$-completeness even when the input graph is restricted to be a planar, bipartite graph of maximum degree three. The vertex subset property $[\rho_{\geq 1}, \sigma_{\leq 1}]$ is shown to share complexity status with $[\rho_{\geq 1}, \sigma_0]$, Independent Dominating sets, in that both minimum and maximum problems are hard while the existence problem is easy. Finally, we give greedy polynomial-time algorithms for solving a class of maximization problems. A natural by-product is the introduction of several new domination-type parameters in graphs. The results given here are a step towards our goal of a complete complexity classification of the problems admitting the characterization.

3.1 Complexity of Old Problems

We use the characterization given in Chapter 2.2. Table 3.1 shows some of the classical

$[L]$-set	Standard terminology	$\exists[L]$	$max[L]$	$min[L]$
$[\rho_{\geq 0}, \sigma_0]$-set	Independent set	P	NPC	P
$[\rho_{\geq 1}, \sigma_{\geq 0}]$-set	Dominating set	P	P	NPC
$[\rho_{\leq 1}, \sigma_0]$-set	Strong Stable set or 2-Packing	P	NPC	P
$[\rho_1, \sigma_0]$-set	Efficient Dominating set or Perfect Code	NPC	NPC	NPC
$[\rho_{\geq 1}, \sigma_0]$-set	Independent Dominating set	P	NPC	NPC
$[\rho_1, \sigma_{\geq 0}]$-set	Perfect Dominating set	P	P	NPC
$[\rho_{\geq 1}, \sigma_{\geq 1}]$-set	Total Dominating set	P	P	NPC
$[\rho_1, \sigma_1]$-set	Total Perfect Dominating set	NPC	NPC	NPC
$[\rho_{\leq 1}, \sigma_{\geq 0}]$-set	Nearly Perfect set	P	P	P
$[\rho_{\leq 1}, \sigma_{\leq 1}]$-set	Total Nearly Perfect set	P	NPC	P
$[\rho_1, \sigma_{\leq 1}]$-set	Weakly Perfect Dominating set	NPC	NPC	NPC
$[\rho_{\geq 0}, \sigma_{\leq n}]$-set	Induced Bounded-Degree subgraph ($n \geq 0$)	P	NPC	P
$[\rho_{\geq n}, \sigma_{\geq 0}]$-set	n-Dominating set ($n \geq 1$)	P	P	NPC
$[\rho_{\geq 0}, \sigma_n]$-set	Induced n-Regular subgraph ($n \geq 0$)	P	NPC	P

Table 3.1: Some vertex subset properties and the complexity of derived problems.

vertex subset properties and also the complexity of derived problems, with P denoting Polytime and NPC denoting $\mathcal{NP}$-Complete. Most of these complexity results are old [34, 23, 32, 12, 24, 33, 22, 44], and others are among the results given in the next section. Table 3.1 can be used as a quick reference guide to the exact definitions of the various properties represented and the complexity of the associated problems.

We are mainly interested in classifying problems admitting the given characterization as $\mathcal{NP}$-complete or as solvable in polynomial time. The objective functions most studied in the past involve minimizing or maximizing the cardinality of the set of selected vertices, and for each entry in Table 3.1, except Nearly Perfect Sets, there is at least one $\mathcal{NP}$-complete problem related to such a parameter. We continue this trend, and the optimization problems we concentrate on are of the form $min[L]$ and $max[L]$. For certain subset properties, such as Perfect Code, it is well known that even deciding if a graph has *any* such set is an $\mathcal{NP}$-complete problem. In the following Lemma we observe several consequences of $\mathcal{NP}$-completeness of an $\exists[L]$ problem.

> **Lemma 3.1** If $\exists[L]$ is $\mathcal{NP}$-complete on a class of graphs C then any decision problems of the form $max[L]$, $min[L]$, $maxM[L]$, $minM[L]$ or $maxL[P]$, $L \subseteq P$ are $\mathcal{NP}$-complete on C. Conversely, if any of the latter problems have a polynomial time algorithm, then so does $\exists[L]$.

Proof. The decision version of $maxM[L]$ takes a graph G and an integer k as input, and asks if G has an $[L]$-set S with $|\{v : state_S(v) \in M\}| \geq k$. Thus, with an algorithm for the decision version of $maxM[L]$, we can decide $\exists[L]$ by a single call of that algorithm providing the integer $k = 0$ as the second part of the input. For $minM[L]$ and $maxL[P]$ problems we would use the integer $k = |V(G)|$ for the input graph G as the second part of the input. □

In particular, Theorems 3.1,3.2,3.3,3.4 and 3.7 can each be combined with Lemma 3.1 to yield corollaries of this kind. We will not state these corollaries explicitly. We observe from Table 3.1 that the vertex subset properties attracting most interest in the past are characterizable by two syntactic states (using the abbreviations) with vertices having zero, one, at least zero, or at least one selected neighbors. Our focus partially continues this trend.

3.2 $\mathcal{NP}$-Completeness Results

We show $\mathcal{NP}$-completeness of several infinite classes of vertex subset problems by reducing from the $\mathcal{NP}$-complete problem Exact 3-Cover (problem [SP2] in [34].)

> **Definition 3.1**Exact 3-Cover (X3C)
>
> Instance: Set U and $T \subseteq \binom{U}{3}$.
>
> Question: $\exists T' \subseteq T$, where T' a partition of U?

We introduce each $\mathcal{NP}$-completeness result by way of a short comparison with the complexity of some related problem from Table 3.1. In contrast to the $\mathcal{NP}$-complete problem of deciding existence of $[\rho_1, \sigma_0]$-sets (Perfect Codes), our first result shows the $\mathcal{NP}$-completeness of certain $\exists[L]$ problems with L containing an infinite number of states.

> **Theorem 3.1** The decision problems $\exists[\rho_{\geq q}, \sigma_0]$ are NP-complete for all $q \in \{2, 3, ...\}$.

Proof. We give a reduction from X3C to $\exists[\rho_{\geq q}, \sigma_0]$ for any $q \in \{2, 3, ...\}$. Given an instance (U, T) of X3C we construct a graph G such that $\exists T' \subseteq T$ with T' a partition of U if and only if G has a $[\rho_{\geq q}, \sigma_0]$-set S. Let $T = \{t_1, ..., t_{|T|}\}$. For each $u \in U$, let $T_u = \{t \in T : u \in t\} = \{t_{u1}, t_{u2}, ..., t_{uk}\}$ be the triples containing u. For each $u \in U$ the graph G will contain a subgraph G_u consisting of a complete graph on vertices $\{x_u, u_{u1}, u_{u2}, ..., u_{uk}\}$ and $q-1$ leaves L_u, each adjacent only to x_u. For each $t_i \in T$ with $t_i = \{u, v, w\}$ we construct a subgraph G_i sharing the vertices u_i, v_i, w_i with G_u, G_v, G_w, respectively, as follows: (case $q = 2$) G_i is a 6-cycle on vertices $A_i \cup B_i$ such that $A_i = \{u_i, v_i, w_i\}$ are mutually non-adjacent; (case $q \geq 3$) G_i is a complete bipartite graph $K_{q,q}$ with partition (A_i, B_i) and with $\{u_i, v_i, w_i\} \subseteq A_i$. This completes the description of G, see Figure 3.1.

Let S be a $[\rho_{\geq q}, \sigma_0]$-set of G. Note that every leaf in L_u must be in S, since ρ_0 and ρ_1 are not legal vertex states. In turn, their common neighbor x_u cannot be in S since σ_1 is not legal. Since $|L_u| = q - 1$ and ρ_{q-1} is not legal at least one other neighbor of x_u, besides its L_u-neighbors, must be in S, i.e. $|\{u_{u1}, u_{u2}, ..., u_{uk}\} \cap S| \geq 1$. But $\{u_{u1}, u_{u2}, ..., u_{uk}\}$ induce a complete graph in G, and σ_0 is the only legal σ-state, so exactly one of these vertices must be in S. Let $u_i \in S$ with $t_i = \{u, v, w\}$. We would want t_i to cover u, v, w and show that indeed we must have $\{u_i, v_i, w_i\} \subseteq S$. Note that these three vertices are all in the same partition A_i of the bipartite graph G_i. We argue first the case $q \geq 3$. No vertex in partition B_i can be in S since already $u_i \in S$ and σ_0 is the only legal σ-state. Moreover, since the neighborhood of any vertex in B_i is exactly A_i and $|A_i| = q$ we must have $A_i \subseteq S$ since ρ_k is not legal for any $k < q$. If $q = 2$ we have G_i a cycle and $u_i \in S$ again forces $A_i \subseteq S$. With this in mind, we have that $T' = \{t_i : A_i \subseteq S\}$ must be an exact 3-cover of U.

For the other direction, if $T' \subseteq T$ is an exact 3-cover of U, it is easy to check that $S = \{v : v \in L_u \wedge u \in U\} \cup \{v : v \in A_i \wedge t_i \in T'\} \cup \{v : v \in B_i \wedge t_i \notin T'\}$ is a $[\rho_{\geq q}, \sigma_0]$-set of G. NP-completeness of the $\exists[\rho_{\geq q}, \sigma_0]$ problem follows, since in polynomial time it is easy to verify a $[\rho_{\geq q}, \sigma_0]$-set and compute the transformation. □

In contrast to $[\rho_{\geq 1}, \sigma_0]$-sets (Independent Dominating sets) which are easily found using a greedy algorithm, our next result shows that $[\rho_{\geq 1}, \sigma_1]$-sets, which we call Dominating Induced Matchings, are difficult to find.

Theorem 3.2 The decision problem $\exists[\rho_{\geq 1}, \sigma_1]$ (Dominating Induced Matching) is NP-complete.

Proof. We again reduce from X3C and adopt all the notation from the proof of Theorem 3.1, constructing gadgets G_u and G_i sharing a vertex u_i if $u \in t_i \in T$. G_u will consist of a complete graph on the vertices $\{x_u, u_{u1}, u_{u2}, ..., u_{uk}\}$ and for each pair $u_i, u_j, i \neq j$ we add three new vertices and edges forming a 5-path from u_i through the new vertices to u_j. See Figure 3.2 which also shows the gadget G_i for $t_i = \{u, v, w\}$. Let S be a $[\rho_{\geq 1}, \sigma_1]$-set in the graph G thus constructed from an instance of X3C. We note right away that for any any vertex $v \in V(G)$ we have $N(v) \cap S \neq \emptyset$ since neither ρ_0 nor σ_0 are legal states. Employing this argument to x_u of the gadget G_u shows that $|\{u_{u1}, u_{u2}, ..., u_{uk}\} \cap S| \geq 1$. Moreover, we cannot have $u_i, u_j \in S$ for $i \neq j$ since the middle vertex on the 5-path from u_i to u_j would have no S-neighbors. Hence, $|\{u_{u1}, u_{u2}, ..., u_{uk}\} \cap S| = 1$. The gadget G_i for a triple $t_i = \{u, v, w\}$ forces either $u_i, v_i, w_i \in S$ or $u_i, v_i, w_i \notin S$, see Figure 3.2. Thus, if we let T' be the triples t_i which have the shared vertices of G_i selected then T' must be an Exact 3-Cover of U. For the other direction of the proof, it is not hard to see from Figure 3.2 that an Exact 3-Cover of the instance (U, T) likewise gives rise to a $[\rho_{\geq 1}, \sigma_1]$-set in G. □

The $\exists[L]$ problem has trivially the affirmative answer if $\rho_0 \in L$. If L contains no ρ-states the $\exists[L]$-problem on G is solved by checking whether for each vertex v we have $\sigma_{deg_G(v)} \in L$. If L contains no σ-states the $\exists[L]$-problem on G is solved by checking whether $\rho_0 \in L$. In light of this, our next theorem gives a complete characterization, up to $\mathcal{P}$ vs. $\mathcal{NP}$, of the complexity of $\exists[L]$ problems when L has a finite number of states. The reduction given is a generalization of a reduction used in [46].

Theorem 3.3 The $\exists[L]$ problem is $\mathcal{NP}$-complete if $\rho_0 \notin L$ and L contains a finite positive number of both ρ-states and σ-states.

Proof. Let $L = \{\rho_{p_1}, \rho_{p_2}, ..., \rho_{p_m}, \sigma_{q_1}, \sigma_{q_2}, ..., \sigma_{q_n}\}$, where $n, m \geq 1$ and p_i, q_i non-negative integers satisfying $0 < p_1 < p_2 < ... < p_m$ and $q_1 < q_2 < ... < q_n$. We reduce from X3C. Given an instance (U, T) of X3C we want a graph G such that G has an $[L]$-set $S \subseteq V(G)$ if and only if $\exists T' \subseteq T$, a partition of U. The gadget for $u_i \in U$ is simply the vertex u_i, which will be shared by gadgets G_t for all triples t with $u_i \in t \in T$. The graph G will be defined by describing the gadgets G_t, one for each $t \in T$. For all $t = \{u_{t1}, u_{t2}, u_{t3}\} \in T$ we construct a graph G_t with private vertices P_t and shared vertices u_{t1}, u_{t2}, u_{t3}, i.e. $V(G_t) = P_t \cup \{u_{t1}, u_{t2}, u_{t3}\}$, having the property:

In the graph G_t, all $S \subseteq V(G_t)$ that assign $\forall v \in P_t$ a state $state_S(v) \in L$ assigns to u_{t1}, u_{t2}, u_{t3} either

(i) $state_S(u_{t1}) = state_S(u_{t2}) = state_S(u_{t3}) = \rho_0$ or

(ii) $state_S(u_{t1}) = state_S(u_{t2}) = state_S(u_{t3}) = \rho_{p_m}$.

Moreover, sets of type (i) and sets of type (ii) should exist for G_t.

Assuming we can construct such G_t, the theorem will follow:

Claim1: $G = \cup_{t \in T} G_t$ has $[L]$-set $S \subseteq V(G) \Leftrightarrow \exists T' \subseteq T$, a partition of U.

($\Leftarrow$:) Note the parts G_t of the graph G share only the vertices representing U. For each $t \in T'$ choose a set $S_t \subseteq V(G_t)$ of type (ii) for G_t. For each $t \notin T'$ choose a set $S_t \subseteq V(G_t)$ of type (i) for G_t. Let $S = \cup_{t \in T} S_t$.

($\Rightarrow$:) For any $[L]$-set S of G we must have $S \cap V(G_t)$ be either a set of type (i) or a set of type (ii) for G_t. This since only G_t contains the vertices P_t, and also $\{w : w \in N(v) \wedge v \in P_t\} \subseteq V(G_t)$. Since $\rho_0 \notin L$, and since a vertex $u \notin S$ can have at most p_m neighbors in S, we must have that $T' = \{t : V(G_t) \cap S$ is a set of type (ii) for $G_t\}$ is a partition of U.

Construction of G_t: Let $V(G_t) = A \cup B \cup X \cup Y \cup Z \cup \{c\} \cup \{u_{t1}, u_{t2}, u_{t3}\}$. See Figure 3.3 for a rough sketch of how these components are connected together. As a preview, we mention that $\{A \cup Y\}$ will be a selected set of type (ii) and $\{B \cup Y\}$ a selected set of type (i) for G_t. X and Y will be such that a selected set must contain all vertices from Y but cannot contain any vertex from X. The vertex c, which cannot be selected, will be connected to enough vertices of Y so that none of its other neighbors, namely $Z \cup \{u_{t1}, u_{t2}, u_{t3}\}$, can be selected. The vertices Z will ensure that either all or none of the neighbors of u_{tk} are selected.

Let $A = A^1 \dot\cup ... \cup A^{p_m}$ and $B = B^1 \cup ... \cup B^{p_m}$ with $A^i = \{a^i_1, ..., a^i_{q_1+1}\}$ and $B^i = \{b^i_1, ..., b^i_{q_1+1}\}$, and let $G[A^i]$, $G[B^i], \forall i$ be complete graphs on $q_1 + 1$ vertices, with no other edges between As or between Bs. Edges connecting vertices of A with vertices of B are restricted to $(a^i_k, b^j_k), \forall i, j, k$. Edges incident with $\{u_{t1}, u_{t2}, u_{t3}\}$ in G_t are restricted to (c, u_{tk}) and $(a^i_1, u_{tk}), \forall i, k$.

Let $\beta = \max\{p_m, q_n\} > 0$ and $\alpha = \lceil \frac{\beta}{(p_1(q_n+1))} \rceil > 0$.

Let $Y = Y^1 \cup ... \cup Y^{p_1 \alpha}$ and $G[Y^i], \forall i$, a complete graph on $q_n + 1$ vertices.

Let $X = \{x_1, x_2, ..., x_{(q_n+1)(\beta+1)\alpha}\}$ with $G[X]$ containing no edges.

We add edges connecting X-vertices with Y-vertices such that each vertex of X gets p_1 neighbors in Y and each vertex of Y gets $\beta + 1$ neighbors in X. This can be done since $|X| = \alpha(q_n + 1)(\beta + 1)$ and $|Y| = \alpha(q_n + 1)p_1$.

The vertex c is connected to p_m vertices of Y, note $|Y| \geq p_m > 0$, and c is also connected to every vertex of $Z \cup \{u_{t1}, u_{t2}, u_{t3}\}$.

It remains to describe the vertices and edges contributed by Z. Let $Z = Z^1 \cup Z^2 \cup Z^3 \cup \{z'\}$ with $Z^k = \{z^k_1, ..., z^k_{p_m}\}$ for $k \in \{1, 2, 3\}$.

The vertex $z^1_i, \forall i$, is connected to a^1_1 and to b^i_1 and has $p_1 - 1$ neighbors in Y.

The vertex $z^2_i, \forall i$, is connected to a^1_1 and to b^i_1 and has $p_m - 1$ neighbors in Y.

The vertex $z^3_i, \forall i$, is connected to a^i_1 and to b^i_1 and has $p_1 - 1$ neighbors in Y.

The vertex z' is connected to $\{a^1_1, ..., a^{p_m}_1, b^1_1, ..., b^{p_m}_1\}$.

This completes the description of G_t.

Claim2: $A \cup Y$ is a set of type (ii) and $B \cup Y$ is a set of type (i) for G_t.

Proof of claim: We consider $A \cup Y$ first. $G[A \cup Y]$ is a collection of p_m copies of K_{q_1+1} for the As and $p_1\alpha$ copies of K_{q_n+1} for the Ys, so $state_{A\cup Y}(a) = \sigma_{q_1}, \forall a \in A$ and $state_{A\cup Y}(y) = \sigma_{q_m}, \forall y \in Y$. Moreover, $\forall x \in X$ we have $N(x) \subseteq Y$ and $|N(x)| = p_1$ so $state_{A\cup Y}(x) = \rho_{p_1}$. For the vertex c we have $N(c) \subseteq Y \cup Z \cup \{u_{t1}, u_{t2}, u_{t3}\}$ and $|N(c) \cap Y| = p_m$, so $state_{A\cup Y}(c) = \rho_{p_m}$. The vertices $z \in Z^1 \cup Z^3$ have $|N(z) \cap \{A \cup Y\}| = p_1$, so $state_{A\cup Y}(z) = \rho_{p_1}$. Similarly, $\forall z \in Z^2$ we have $|N(z) \cap \{A \cup Y\}| = p_m$, so $state_{A\cup Y}(z) = \rho_{p_m}$. The vertex z' has $N(z') \subseteq A \cup B$ and $|N(z) \cap A| = p_m$, so $state_{A\cup Y}(z') = \rho_{p_m}$. So far, the argument for $B \cup Y$ being a set of type (i) can be obtained from the above by replacing B for A and vice-versa.

Since $\forall b \in B$, $N(b) \subseteq A \cup Z$ and $|N(b) \cap A| = p_m$, we have $state_{A\cup Y}(b) = \rho_{p_m}$. Similarly, $\forall a \in A$ we have $N(a) \subseteq B \cup Z \cup \{u_{t1}, u_{t2}, u_{t3}\}$ and $|N(a) \cap B| = p_m$, so $state_{B\cup Y}(a) = \rho_{p_m}$.

What remains is the argument for the vertices $\{u_{t1}, u_{t2}, u_{t3}\}$. We have for $k \in \{1, 2, 3\}$, $N(u_{tk}) = \{a^1_1, ..., a^{p_m}_1\}$, so $state_{A\cup Y}(u_{tk}) = \rho_{p_m}$ and $state_{B\cup Y}(u_{tk}) = \rho_0$ so that $A \cup Y$ is a set of type (ii) and $B \cup Y$ is a set of type (i), completing the proof of the claim.

Claim3: For any $S_t \subseteq V(G_t)$ which assigns $\forall w \in V(G_t) \setminus \{u_{t1}, u_{t2}, u_{t3}\}$ a state $state_{S_t}(w) \in L$, we have $Y \subseteq S_t$ and also $(Z \cup \{u_{t1}, u_{t2}, u_{t3}\}) \cap S_t = \emptyset$.

Proof of claim: $\forall y \in Y$ we have $|N(y) \cap X| = \beta + 1 > \max\{p_m, q_n\}$, so $\exists x \in N(y) : x \notin S_t$. But $|N(x) \cap Y| = p_1$, so $state_{S_t}(x) = \rho_{p_1}$ and $y \in S_t$. Since $|N(y) \cap Y| = q_n$ we must have $state_{S_t}(y) = \sigma_{q_n}$. Since $|N(c) \cap Y| = p_m$ we must have $state_{S_t}(c) = \rho_{p_m}$ and $(Z \cup \{u_{t1}, u_{t2}, u_{t3}\}) \cap S_t = (N(c) \setminus Y) \cap S_t = \emptyset$, completing the proof of the claim.

Claim4: For any $S_t \subseteq V(G_t)$ which assigns $\forall w \in V(G_t) \setminus \{u_{t1}, u_{t2}, u_{t3}\}$ a state $state_{S_t}(w) \in L$, we have either $a^i_1 \in S_t, 1 \leq i \leq p_m$ or $a^i_1 \notin S_t, 1 \leq i \leq p_m$.

Proof of claim: From Claim3 we have $Z \cap S_t = \emptyset$ and $Y \subseteq S_t$. In particular, $state_{S_t}(z^1_i) \in \{\rho_{p_1}, \rho_{p_1+1}\}$, similarly $state_{S_t}(z^2_i) \in \{\rho_{p_m-1}, \rho_{p_m}\}$ and $state_{S_t}(z^3_i) \in \{\rho_{p_1}, \rho_{p_1+1}\}$. In turn, we consider the two cases $a^1_1 \in S_t$ and $a^1_1 \notin S_t$. $a^1_1 \in S_t$ gives $state_{S_t}(z^2_i) = \rho_{p_m}, \forall i$, so $b^i_1 \notin S_t, \forall i$. This in turn gives $state_{S_t}(z^3_i) = \rho_{p_1}$ so $a^i_1 \in S_t, \forall i$, completing the first case. $a_1^1 \notin S_t$ implies $b_1^i \in S_t, \forall i$ so that $state_{S_t}(z_i^1) = \rho_{p_1}, \forall i$. This in turn gives $state_{S_t}(z') = \rho_{p_m}$ so $a_1^i \notin S_t, \forall i$, completing the proof of the claim.

Each of $\{u_{t1}, u_{t2}, u_{t3}\}$ is adjacent to exactly $\{a_1^1, ..., a_1^{p_m}\}$ and by Claim3 cannot be in S_t. Hence, Claim4 actually shows that any $S_t \subseteq V(G_t)$ which assigns $\forall w \in V(G_t) \setminus \{u_{t1}, u_{t2}, u_{t3}\}$ a state $state_{S_t}(w) \in L$ has either
(i) $state_{S_t}(u_{t1}) = state_{S_t}(u_{t2}) = state_{S_t}(u_{t3}) = \rho_0$, or
(ii) $state_{S_t}(u_{t1}) = state_{S_t}(u_{t2}) = state_{S_t}(u_{t3}) = \rho_{p_m}$.
Thus G_t has the claimed properties and the theorem follows. □

As our next theorem shows, some of these decision problems are $\mathcal{NP}$-complete even for very restricted classes of graphs. The reduction we use is a simple special case of the one just given, and uses the $\mathcal{NP}$-complete problem Planar 3-Dimensional Matching (P3DM). A similar reduction is used in [32].

Definition 3.2 3-Dimensional Matching (3DM) [SP1]

Instance: Disjoint sets U_1, U_2, U_3 with $U = U_1 \cup U_2 \cup U_3$ and $T \subseteq U_1 \times U_2 \times U_3$.

Question: $\exists T' \subseteq T$, where T' a partition of U?

With an instance I of 3DM, we associate the bipartite graph G_I where $V(G_I) = U \cup T$ and $E(G_I) = \{(u,t) : u \in U \wedge u \in t \in T\}$. In [29] it is shown that Planar 3-Dimensional Matching, 3DM restricted to instances where G_I is planar, is still $\mathcal{NP}$-complete.

Theorem 3.4 The problem of deciding whether a planar bipartite graph of maximum degree three has any $[\rho_1, \sigma_1]$-set (Total Perfect Dominating Set) is $\mathcal{NP}$-complete.

Proof. Given an instance I of P3DM, we construct a graph G having a $[\rho_1, \sigma_1]$-set if and only if $\exists T' \subseteq T$, a partition of U. Let G be the graph G_I augmented by adding, for each $t \in T$, the vertices a_t and b_t, and edges connecting a_t to both t and b_t. Since this reduction does not distinguish between the sets U_1, U_2, U_3, the instance I can be viewed as an instance of X3C, and the argument that G has a $[\rho_1, \sigma_1]$-set if and only if $\exists T' \subseteq T$, a partition of U, is left out since it is in easy analogy with the argument used for the previous theorem.

Note that G_I and G are both planar bipartite graphs. We next show an easy transformation of a graph G having a vertex of degree larger than three to a graph G' with the following properties:
(i) if G planar and bipartite then G' planar and bipartite,
(ii) $\Sigma_{\{v:deg_G(v)\geq 4\}} deg_G(v) > \Sigma_{\{v:deg_{G'}(v)\geq 4\}} deg_{G'}(v)$
(iii) G has a $[\rho_1, \sigma_1]$-set if and only if G' has a $[\rho_1, \sigma_1]$-set.

Hence, applying such a polytime transformation repeatedly, starting with G, until the resulting graph has no vertices of degree larger than three, yields a graph proving the theorem.

We define the transformation by describing the resulting graph G'. Let v be a distinguished vertex of G with $N_G(v) = \{v_1, v_2, ..., v_k\}$ and $k \geq 4$. Let G' have vertices $V(G') = V(G) \cup \{w, x, y, z\}$ and edges

$$E(G') = E(G) \setminus \{(v_1, v), (v_2, v)\} \cup \{(v_1, w), (v_2, w), (w, x), (x, y), (y, z), (z, v)\}$$

See Figure 3.4. Note the transformation is local, with changes only to the neighborhoods of v_1, v_2 and v.

We prove the stated properties of the transformation:

(i) Planarity is obviously preserved. If A, B is an appropriate bipartition of $V(G)$ then w.l.o.g. we must have $v \in A$, $N(v) \subseteq B$ so that $A \cup \{w, y\}$ and $B \cup \{x, z\}$ forms an appropriate bipartition of $V(G')$. (ii) The new vertices all have degree less than 4, whereas the degree of v decreases to $k-1$. (iii) Let S and S' be $[\rho_1, \sigma_1]$-sets in G and G', respectively. Note that $\{w, x, y, z, v\}$ induces a 5-path in G' so there are 4 possibilities for $\{w, x, y, z, v\} \cap S'$, namely $\{y, z\}, \{w, z, v\}, \{w, x, v\}$ and $\{x, y\}$. We similarly split the possibilities for choice of S into 4 classes, namely

$|\{v_1, v_2\} \cap S| = 1 \wedge v \notin S \wedge |\{v_3, ..., v_k\} \cap S| = 0$,
$|\{v_1, v_2\} \cap S| = 1 \wedge v \in S \wedge |\{v_3, ..., v_k\} \cap S| = 0$,
$|\{v_1, v_2\} \cap S| = 0 \wedge v \in S \wedge |\{v_3, ..., v_k\} \cap S| = 1$,
$|\{v_1, v_2\} \cap S| = 0 \wedge v \notin S \wedge |\{v_3, ..., v_k\} \cap S| = 1$.

It is easy to check that the 4 possibilities for choice of S' have, in the order given, characterizations in terms of effect on v and $N(v)$ which are identical to those just given for S, and indeed property (iii) holds. □

To our knowledge, the complexity of problems defined over Total Perfect Dominating Sets in graphs, had not been studied previously [23].

Combining Lemma 3.1 with Theorem 3.4 gives the $\mathcal{NP}$-completeness on planar bipartite graphs of maximum degree three of the problem $max\{\rho_1, \sigma_1\}[\rho_{\geq 0}, \sigma_{\geq 0}]$, which we call Total Efficiency. This problem arises in communication networks, if we assume that a communication round has two time-disjoint phases, *send* and *receive*, and that a processor receives a message whenever it has a single sending neighbor. The maximum number of processing elements that can receive a message in one communication round is the Total Efficiency of the graph underlying the network topology.

The following strong result is due to Kratochvíl.

> **Theorem 3.5**[46] The problem of deciding whether a planar 3-regular graph has a $[\rho_1, \sigma_0]$-set (perfect code) is $\mathcal{NP}$-complete.

We state the implications of this result for some other problems admitting our characterization.

Corollary 3.1 Any decision problem of the form $min[L]$ with $\rho_0 \notin L$ and $\{\rho_1, \sigma_0\} \subseteq L$ is $\mathcal{NP}$-complete on planar 3-regular graphs.

Proof. Let G be a planar 3-regular graph. We show that G has a perfect code if and only if the value of $min[L]$ on G is $|V(G)|/4$. Since every vertex of G has degree 3, a perfect code of G has size $|V(G)|/4$ and is clearly a dominating set. Moreover, a dominating set of G which is not a perfect code will have more than $|V(G)|/4$ vertices. An $[L]$-set in G is a dominating set since ρ_0 is not legal and it could be a perfect code since ρ_1 and σ_0 are legal. The corollary follows. $\square$

While every graph has an Independent Dominating Set ($[\rho_{\geq 1}, \sigma_0]$-set), that can be easily found by a greedy algorithm, it is well-known that both minimizing and maximizing the size of such a set is $\mathcal{NP}$-hard. Our next result shows another vertex subset property with this complexity classification.

Theorem 3.6 The decision problems $min[\rho_{\geq 1}, \sigma_{\leq 1}]$ and $max[\rho_{\geq 1}, \sigma_{\leq 1}]$ are both $\mathcal{NP}$-complete, while $\exists[\rho_{\geq 1}, \sigma_{\leq 1}]$ can be solved in linear time.

Proof. Any graph has a $[\rho_{\geq 1}, \sigma_{\leq 1}]$-set, take for example a $[\rho_{\geq 1}, \sigma_0]$-set, easily found in $\mathcal{O}(|E(G)| + |V(G)|)$ by a greedy algorithm. NP-completeness of $min[\rho_{\geq 1}, \sigma_{\leq 1}]$ follows from Corollary 1. We show $\mathcal{NP}$-completeness of $max[\rho_{\geq 1}, \sigma_{\leq 1}]$ by reduction from $max[\rho_{\geq 1}, \sigma_0]$. Given a graph G, construct the graph G' with $V(G') = \{u_1, u_2 : u \in V(G)\}$ and $E(G') = \{(u_1, u_2) : u \in V(G)\} \cup \{(u_1, v_1), (u_2, v_2), (u_1, v_2), (u_2, v_1) : (u, v) \in E(G)\}$, see Figure 3.5. Let S be a maximum-size $[\rho_{\geq 1}, \sigma_0]$-set in G and let S' be a maximum-size $[\rho_{\geq 1}, \sigma_{\leq 1}]$-set in G'. We show that $2|S| = |S'|$. Let A be $[\rho_{\geq 1}, \sigma_0]$ in G. Then $A' = \{u_1, u_2 : u \in S\}$ is $[\rho_{\geq 1}, \sigma_{\leq 1}]$ in G'. We have $2|A| = |A'|$, so this shows that $2|S| \leq |S'|$. Let B' be $[\rho_{\geq 1}, \sigma_{\leq 1}]$ in G', with $C = \{(u_i, v_j) \in E(G') : \{u_i, v_j\} \subseteq B'\}$, the edges of $G'[B']$. Choose one endpoint of each edge from C and call this set of vertices D. Define $B = \{v \in V(G) : state_{B'}(v_1) = \sigma_0 \vee state_{B'}(v_2) = \sigma_0 \vee v_1 \in D \vee v_2 \in D\}$. Since we have removed one endpoint from each edge of $G'[B']$ it is clear that B is an independent set in G and $2|B| \geq |B'|$. In our notation, B is $[\rho_{\geq 0}, \sigma_0]$ in G, and can be greedily augmented to a $[\rho_{\geq 1}, \sigma_0]$-set, which shows that $2|S| \geq |S'|$. The transformation is easily computed in polynomial time, and the theorem follows. $\square$

Minimization problems of the form $min[L]$ have the empty vertex subset as solution if $\rho_0 \in L$. Similarly, if L has no σ-states then the empty vertex subset is the only possible solution. If L has no ρ-states then the only possible $[L]$-set in a graph G is $V(G)$ which is checked by degree computation as described earlier. A $min[L]$ problem where L does not satisfy any of the above is asking for a minimum-size dominating set S of a certain kind. We have reason to believe that finding such a set is, in general, $\mathcal{NP}$-hard.

Conjecture 3.1 Assuming $\mathcal{P} \neq \mathcal{NP}$ the decision problem $min[L]$ is $\mathcal{NP}$-complete if and only if $\rho_0 \notin L$ and L contains both some ρ-state and some σ-state.

3.3 Efficient Algorithms

We now turn to vertex subset problems which have an easy solution algorithm, and focus our attention on optimization problems. Based on Lemma 3.1 such results have as corollaries the polynomial-time solvability of the associated existence problems. For $min[L]$ problems, we believe the argument given above for one direction of Conjecture 3.1 constitute the only polynomial-time cases. For $max[L]$ problems we have the following.

Theorem 3.7 The problem $max[L]$ is solvable in polynomial time by a greedy algorithm if $\sigma_{\geq k}$ is the only σ-state in L and either (i), (ii), (iii) or (iv) holds

(i) $\rho_1, \rho_2 ..., \rho_{k-1} \in L$

(ii) $\rho_0, \rho_1, ..., \rho_{k-1} \notin L$

(iii) $\rho_{\geq h}$ is the only ρ-state in L, for some h

(iv) ρ_0 and $\rho_{\geq h}$ are the only ρ-states in L, for some h

Proof. We give two greedy algorithms, named ALG1 and ALG2, with input a graph G and output a set achieving $max[L]$ for G, if any $[L]$-set exists. ALG2 is used in case (iv) when $h \geq 2$ in which case there is a crucial gap in the legal ρ-states while ALG1 is used in the remaining cases. The algorithms use data structures $B\sigma$,$B\rho$ of type *set*.

ALG1(G)
$B\sigma,\, B\rho := V(G),\, \emptyset$;
while (I: $\exists v \in B\sigma : |N(v) \cap B\sigma| < k$) do $B\sigma,\, B\rho := B\sigma \setminus \{v\},\, B\rho \cup \{v\}$;
if ($\exists v \in B\rho : state_{B\sigma}(v) \notin L$) then output($\nexists[L]$-set) else output($B\sigma$);

ALG2(G)
$B\sigma,\, B\rho := V(G),\, \emptyset$;
while (I: $\exists v \in B\sigma : |N(v) \cap B\sigma| < k$) or (II: $\exists w \in B\rho : |N(w) \cap B\sigma| < h$) do
    Case I : $B\sigma,\, B\rho := B\sigma \setminus \{v\},\, B\rho \cup \{v\}$;
    Case II: $B\sigma,\, B\rho := B\sigma \setminus \{N(w) \cap B\sigma\},\, B\rho \cup \{N(w) \cap B\sigma\} \setminus \{w\}$;
output($B\sigma$);

We first prove correctness, for both algorithms, of the loop invariant: "A vertex $v \notin B\sigma$ cannot be a member of any $[L]$-set S of G." The loop invariant is true initially since $B\sigma = V(G)$. Let $B\sigma$ and $B\sigma'$ be the values before and after an iteration of the loop. From the loop invariant we have $S \subseteq B\sigma$ and show that $S \subseteq B\sigma'$.

Case I (both algorithms): $B\sigma \setminus B\sigma' = \{v\}$ and $v \in B\sigma : |N(v) \cap B\sigma| < k$. Since $\sigma_{\geq k}$ is the only σ-state in L, $S \subseteq B\sigma$ cannot contain v.

Case II (ALG2 only, i.e. ρ_0 and $\rho_{\geq h}$ the only legal ρ-states): $v \in B\sigma \setminus B\sigma'$ and $\exists w : v \in N(w)$ where $w \in B\rho$ and $|N(w) \cap B\sigma| < h$. When a vertex is added to $B\rho$ it is also removed from the non-growing set $B\sigma$ so that $B\rho \cap B\sigma = \emptyset$ and in particular $w \notin S$. Since $\rho_1, \rho_2, ..., \rho_{h-1} \notin L$ we must have $state_S(w) = \rho_0$ so that $N(w) \cap S = \emptyset$. Since $v \in N(w)$ this completes the proof of the loop invariant.

At termination of both algorithms all vertices in $B\sigma = S$ have at least k neighbors in S. We first argue correctness of ALG1. At termination of ALG1 all vertices not in S (in $B\rho$) have less than k neighbors in S, so if for some $v \in B\rho$ we have $state_S(v) = \rho_i \notin L$ there cannot be any $[L]$-set in G since for no $j < i$ is $\rho_j \in L$. However, if such a vertex does not exist S is a maximum-size $[L]$-set, proving correctness of ALG1. At termination of ALG2 all vertices not in $B\sigma = S$ have either at least h neighbors in S (these vertices are in $B\rho$) or no neighbors in S. Since ρ_0 and $\rho_{\geq h}$ are both legal ρ-states we have S a maximum $[L]$-set, and ALG2 is correct. $\square$

Chapter 4

Complexity of H-Covering Problems

Efficient algorithms for certain graph covering problems are designed according to two basic techniques. The first one is a reduction to the 2-SAT problem. The second technique exploits necessary and sufficient conditions for the existence of regular factors in graphs. For other infinite classes of graph covering problems we derive $\mathcal{NP}$-completeness results by reductions from graph coloring problems. We illustrate this methodology by classifying all graph covering problems defined by simple graphs with at most 6 vertices.

4.1 Motivation and Overview

For a fixed graph H, the H-cover problem admits a graph G as input and asks about the existence of a "local isomorphism": a labeling of vertices of G by vertices of H so that the label set of the neighborhood of every $v \in V(G)$ is equal to the neighborhood (in H) of the label of v. We trace this concept to Biggs' construction of highly symmetric graphs in [15], and to Angluin's discussion of "local knowledge" in distributed computing environments in [3]. More recently, Abello *et al.* [1] raised the question of computational complexity of H-cover problems, noting that there are both polynomial-time solvable (*easy*) and $\mathcal{NP}$-complete (*difficult*) versions of this problem for different graphs H. Our own interest in the subject comes from viewing H-covering as a vertex partitioning problem. Using the adjacency matrix $D_{|V(H)|}$ of H as a degree constraint matrix (with singleton entries), the H-cover problem is simply the $\exists D_{|V(H)|}$ problem. A related question of complexity of H-coloring, where the degree constraint matrix is obtained from the above matrix by replacing $\{1\}$-entries with $\mathbb{N}$, has been resolved by Hell and Nešetřil [42] who completely classified graphs for which the problem is easy and those for which it is difficult. Our goal is to extend our current findings to achieve a complete classification also for the complexity of the H-cover problem.

We develop a methodology that is useful in analyzing the complexity of graph

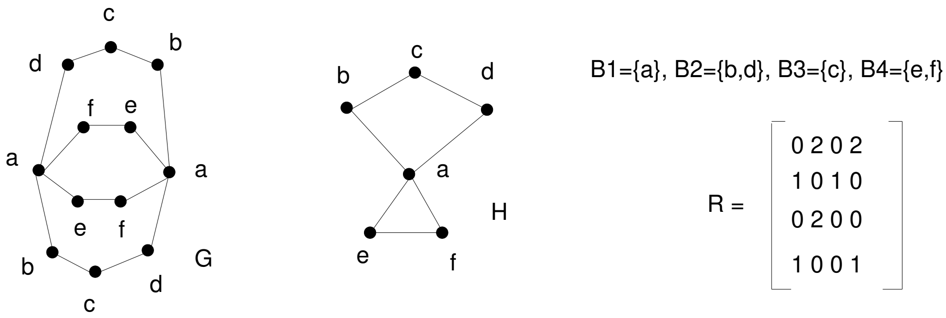


Figure 4.1: G labeled by a covering projection of H, their common degree refinement R and the degree partition of H.

covering problems. In designing efficient algorithms that solve easy graph covering problems, we reduce those problems to regular factorization problems and/or to the 2-SAT problem. We introduce these tools and present the corresponding results. To prove $\mathcal{NP}$-completeness of the difficult graph covering problems, we use polynomial time reductions from known $\mathcal{NP}$-complete restrictions of vertex, edge and H-coloring problems and also reductions between covering problems. These last reductions are based on properties of the automorphism groups of the relevant graphs. We set up a paradigm to construct such reductions and then present our findings. At the end of this chapter, we give a catalogue of the complexity of the covering problem for all simple graphs with at most 6 vertices. There are 208 such graphs, with 81 having non-trivial polytime solution algorithms and 36 being NP-complete (the remaining graphs defining trivial covering problems).

We give some useful definitions. The H-cover (decision) problem on an input graph G has the affirmative answer iff there is a function (called *covering projection*) $f : V(G) \to V(H)$ which preserves the identity of the neighborhood of any vertex v of G, $\{f(u)|u \in N_G(v)\} = N_H(f(v))$ with $deg_G(v) = deg_H(f(v))$. Fixing the graph H, and allowing any graph G as the input, one can pose the question: "Does G cover H?" The computational complexity of this problem, called the H-*cover* problem for the particular graph H, is the subject of this chapter.

The *degree partition* of a graph is the partition of its vertices into the minimum number of *blocks* $B_1, \ldots, B_t$ for which there are constants r_{ij} such that for each i, j $(1 \leq i, j \leq t)$ each vertex in B_i is adjacent to exactly r_{ij} vertices in B_j. The $t \times t$ matrix R $(R[i,j] = r_{ij})$ is called the *degree refinement*.

The degree partition and degree refinement of a graph are easily computed by a stepwise refinement procedure. Start with vertices partitioned according to degree and keep refining the partition until any two nodes in the same block have the same number of neighbors in any other given block. See Figure 4.1 for an example. Graph coverings are related to degree partitions and degree refinements (see, for instance, Leighton [52]):

Fact 4.1 If f is a covering projection of H by G then H and G have the same degree refinement and have degree partitions $B_1, B_2, ..., B_t$ and $B'_1, B'_2, .., B'_t$ so that for every $v \in B'_i$ we have $f(v) \in B_i$, $i = 1, 2, ..., t$.

Without loss of generality, we will consider only connected graphs, because of the following observations (whose proofs are left to the reader.)

Fact 4.2 Given a connected graph H, a graph G covers H if and only if every connected component of G covers H.

Fact 4.3 For a disconnected graph H, the H-cover problem is polynomially solvable if and only if the H_i-cover problem is polynomially solvable for every connected component H_i of H.

Fact 4.4 For a disconnected graph H, the H-cover problem is $\mathcal{NP}$-complete if for some connected component H_i of H the H_i-cover problem is $\mathcal{NP}$-complete.

As an appendix to this chapter, we list every connected, simple graph H on at most six vertices and at least two cycles, showing the complexity of the H-covering problem. Covering of simple graphs with at most one cycle is easy by Fact 4.7, stated below.

4.2 Efficient Algorithms

For a given graph G and a fixed graph H, it is easy to compare degree partitions and degree refinements in polynomial time. Surprisingly, for many graphs H, the necessary condition for the existence of a covering given by Fact 4.1 is also sufficient. For many other graphs H (including some infinite classes of graphs), for which those conditions are not sufficient, we are able to design an efficient solution algorithm paradigm by constructing an equivalent instance of the 2-SAT problem, and/or by reducing to a factorization problem in a regular graph. Before we present these results, we observe some fairly obvious facts about trees and unicyclic graphs.

Fact 4.5 A graph G covers a given tree H if and only G is isomorphic to H.

Fact 4.6 A graph G covers a cycle H if and only if G is a cycle with the length that is a multiple of the length of H.

The two preceding facts imply indirectly the following observation.

Fact 4.7 For a graph H with at most one cycle, the H-cover problem is solvable in polynomial time.

4.2.1 Reductions to 2-Satisfiability

The 2-SAT problem where clauses have at most two variables is solvable in polynomial time. We reduce a class of H-covering problems to an instance of the 2-SAT problem.

> **Theorem 4.1** The H-COVER problem is solvable in polynomial time if every block of the degree partition of H contains at most two vertices.

Proof. Denote the vertices of the i-th block B_i of H by L_i, R_i (or L_i only, if B_i is a singleton). Suppose that G has the same degree refinement as H and its degree partition is $B'_1, B'_2, \ldots, B'_t$, where the blocks are numbered so that every covering projection sends B'_i onto B_i, $1 \leq i \leq t$. This structure of G can be checked in polynomial time, and G does not cover H unless it satisfies these assumptions.

The crucial part of the algorithm is to decide which vertices of B'_i should map onto L_i and which onto R_i. This can be done via 2-SAT. For every vertex u of G, introduce a variable x_u. In a truth assignment ϕ, these variables would encode

$$\phi(x_u) = \begin{cases} \mathsf{true} & \text{if } f(u) = L_i \\ \mathsf{false} & \text{if } f(u) = R_i \end{cases} \tag{4.1}$$

for a corresponding covering projection f (here i is such that $u \in B'_i$). We construct a formula Φ as a conjuntion of the following subformulas:

1. (x_u) for every $u \in B'_i$ such that B_i is a singleton;
2. $(x_u \vee x_v) \wedge (\neg x_u \vee \neg x_v)$ for any pair of adjacent vertices u, v which belong to the same block B'_i (i.e., $L_iR_i \in E(H)$);
3. $(x_u \vee \neg x_v) \wedge (\neg x_u \vee x_v)$ if u and v belong to distinct blocks (say $u \in B'_i$ and $v \in B'_j$) and there are exactly the two edges L_iL_j, R_iR_j between B_i and B_j in H;
4. $(x_u \vee x_v) \wedge (\neg x_u \vee \neg x_v)$ if u and v belong to distinct blocks (say $u \in B'_i$ and $v \in B'_j$) and there are exactly the two edges L_iR_j, R_iL_j between B_i and B_j in H;
5. $(x_w \vee x_v) \wedge (\neg x_w \vee \neg x_v)$ if v and w belong to the same block (say B'_j) and are both adjacent to u which belongs to a block (say B'_i) such that $L_iL_j, L_iR_j \in E(H)$.

Note that in case 2, every $u \in B'_i$ has exactly one neighbor v in the same block, in cases 3 and 4, every $u \in B'_i$ has exactly one neighbor $v \in B'_j$, and in case 5, every $u \in B'_i$ has exactly two neighbors $v, w \in B'_j$.

It is clear that Φ is satisfiable if and only if f defined by (4.1) is a covering projection from G onto H. The clauses derived from 2 guarantee, if $L_iR_i \in E(H)$, that every vertex mapped on L_i has a neighbor which maps onto R_i and vice versa, the clauses from 3-5 control adjacencies to vertices from different blocks, and the technical clauses from 1 control the singletons. □

4.2.2 Reductions to factorization

A spanning subgraph H of a graph G is a *k-factor* if all vertices of H have degree k. When $k = 1$, the 1-factor is often referred to as *perfect matching*. The existence of perfect matchings in bipartite graphs is a subject of the celebrated König-Hall theorem. A graph G is *k-factorable* if its edges can be partitioned into k-factors. An application of the Hall-König marriage theorem states that a regular bipartite graph is 1-factorable ([37],[49]). We will use this fact to show that the obvious necessary conditions are also sufficient for a class of graph covering problems.

> **Theorem 4.2** Let H be a graph with all but two vertices of degree 2, all of them lying on paths connecting the two vertices of degree $k > 2$. Then a graph G covers H if and only if H and G have the same degree refinement and the multigraph obtained from G by replacing the paths between vertices of degree k by edges is bipartite. It follows that the H-cover problem is solvable in polynoimal time.

Proof. The 'only if' part of the statement is obvious. For the 'if' part, note first that since G is connected, the bipartition of its degree k vertices into V_1, V_2 is unique. Denote the vertices of degree k in H by v_1, v_2, and let the paths between them have lengths $n_1 < n_2 < \ldots < n_m$, with exactly k_i paths of length n_i. Number the paths of length n_i from 1 to k_i. For every i, consider an auxiliary multigraph G_i with vertex set $V_1 \cup V_2$ and edges being in one-to-one correspondence with the paths of length n_i between the vertices of $V_1 \cup V_2$ in G. Since degree refinements of G and H are identical, G_i is k_i-regular bipartite. It is therefore 1-factorable, which means that its edges can be colored by k_i colors, say $1, 2, \ldots, k_i$, so that every vertex is incident to exactly one edge of each color. We then define the covering projection by

$f(x) = v_i$ if $x \in V_i, i = 1, 2$, and otherwise

$f(x) = u$ where u is a vertex of degree 2 on the j^{th} path of length n_i which leads from v_1 to v_2 and x is the corresponding vertex on the path in G (from a vertex in V_1 to a vertex in V_2) that is represented by an edge colored by color j in the auxiliary multigraph G_i. □

If the H-covering problem is easy one may ask for what supergraphs of H the covering problem remains easy. In a forthcoming paper we will treat this question in detail. Presently, to encompass all 6-vertex graphs, we mention that we can safely add a degree 1 vertex to a graph falling under Theorem 4.2.

A classical result of Petersen [57] states that any $2k$-regular graph is 2-factorable. We will use this fact to show that the obvious necessary conditions are also sufficient for a class of graph covering problems.

> **Theorem 4.3** Let H be a graph with all but one vertex of degree 2. Then the H-cover problem is solvable in polynomial time, and a graph G covers H if and only if its degree refinement is the same as the degree refinement of H.

Proof. The structure of H is such that it contains one vertex, say A, of degree $2k$ and all other vertices lie on cycles which pass through A. Let the lengths of the cycles be $n_1 < n_2 < \ldots n_m$ and let there be k_i cycles of length n_i, $i = 1, 2, \ldots, m$.

The obvious necessary condition for a graph G to cover H is for G to contain only vertices of degree 2 and $2k$, and for every $i = 1, 2, \ldots, m$, every vertex of degree $2k$ is an endpoint of exactly $2k_i$ paths of length n_i which contain only vertices of degree 2 and every such path is between degree $2k$ vertices (possibly the same vertex). This is just an explicit reformulation of the fact that G has the same degree refinement as H. We will show that this obvious necessary condition is also sufficient.

It suffices to consider only paths of the same length, say n_i. Consider a multigraph G' whose vertex set are the vertices of degree $2k$ in G, and edges correspond to paths of length n_i. This graph is $2k_i$-regular, and hence 2-factorable [57]. Let E'_j, $j = 1, 2, \ldots, k_i$, be the edge sets of k_i disjoint 2-factors. Each such E'_j is a disjoint union of cycles, which in the original graph G correspond to cycles formed by paths of length n_i. These paths of G must map to paths $P_1, P_2, \ldots, P_{k_i}$ of H, with P_j having vertices $A_{j1}, A_{j2}, \ldots, A_{jn_i}$. In fact, the paths in G represented by a 2-factor E'_j can all be mapped onto the same path P_j in H. If $x_1, x_{11}, \ldots, x_{1n_i}, x_2, x_{21}, \ldots, x_{2n_i}, \ldots, x_r, x_{r1}, \ldots, x_{rn_i}$ is such a cycle (with $x_1, x_2, \ldots, x_r$ being its vertices of degree $2k$), then the vertices x_{ab} will map onto A_{jb}, $1 \leq a \leq r, 1 \leq b \leq n_i$. $\square$

Theorems 4.2 and 4.3 can be unified in the following general statement, which is again an example of the 'obvious necessary conditions are also sufficient' scheme. The proof, which we omit, is more or less a confluence of the proofs of Theorems 4.2 and 4.3.

Theorem 4.4 Let H be a graph with all but two vertices of degree 2 and let these two vertices of higher degree be L and R. Further suppose that for every $i > 1$, L belongs to l_i cycles of length i, R belongs to r_i cycles of length i and there are m_i paths of length i joining L and R. If

a) there is an i such that $l_i \neq r_i$, or

b) $l_i m_i = 0$ for every i,

then H-COVER is solvable in polynomial time, and a graph G covers H if and only if it has the same degree refinement as H and, in case b), if the vertices of degree > 2 in G can be partitioned into classes U and V so that every path of length i such that $m_i \neq 0$ connects a vertex from U to a vertex from V, and every path of length i such that $l_i \neq 0$ either connects vertices from U or vertices from V.

Let us state without proof that in all remaining cases, i.e., when $l_i = r_i$ for all i and there is an i_0 such that $l_0 \neq 0$ and $m_0 \neq 0$, the H-COVER problem is $\mathcal{NP}$-complete.

Fact 4.7, Theorems 4.1, 4.2 and Theorem 4.3 encompass all but three graphs of at most 6 vertices for which the covering problem is easy. One of these graphs is a particularly easy case of Theorem 4.4. The covering problems for the two remaining graphs, which will be treated in detail in a forthcoming paper, are solved by a modification of the 2-SAT method.

4.3 $\mathcal{NP}$-Completeness

For any graph H the H-cover problem is in $\mathcal{NP}$. We will show $\mathcal{NP}$-completeness of H-cover problems for several infinite classes of graphs. We first mention an earlier result.

Theorem 4.5 [46] For every $k \geq 4$, the K_k-cover problem is $\mathcal{NP}$-complete.

4.3.1 Reductions from Coloring Problems

In this section we reduce from $\mathcal{NP}$-complete problems: edge coloring, vertex coloring and H-coloring. The *H-coloring* problem asks for the existence of a labeling of vertices of G by vertices of H, $h : V(G) \to V(H)$, which preserves adjacencies, $uv \in E(G) \Rightarrow h(u)h(v) \in E(H)$. This problem is easy if H is bipartite but $\mathcal{NP}$-complete otherwise [42]. The *vertex k-coloring* problem is equivalent to the K_k-coloring problem. The *edge k-coloring* problem asks if each edge of a graph can be assigned one of k colors so that no two edges incident with the same vertex are assigned the same color. Edge 3-coloring of cubic graphs is $\mathcal{NP}$-complete [43]. The following observation is used in our reductions.

Fact 4.8 If G covers H by $f : V(G) \to V(H)$ and $\pi \in Aut(H)$ then $\pi \circ f$ (the composition of f and π) is also a covering projection of H by G.

The reductions are by vertex and edge gadget construction, providing a graph G' which covers H if and only if a given graph G can be colored appropriately. The general outline of the reductions is as follows:

1. Define vertex gadget for a vertex $v \in V(G)$ by a subgraph of a cover of H, with $deg_G(v)$ 'port's to be used for edge gadgets (edge-coloring requires covers of H with distinct projections for each port and automorphisms of H that allow any permutation of the ports as distinct covers by Fact 4.8, whereas for k-vertex coloring we need equivalent projections for each port and k distinct covers)

2. Define edge gadget connecting two ports by a subgraph of a cover of H, so that the 'only if' direction of the reduction is met (for edge-coloring (vertex coloring) this amounts to ensuring that the two ports must cover H equivalently (distinctly)).

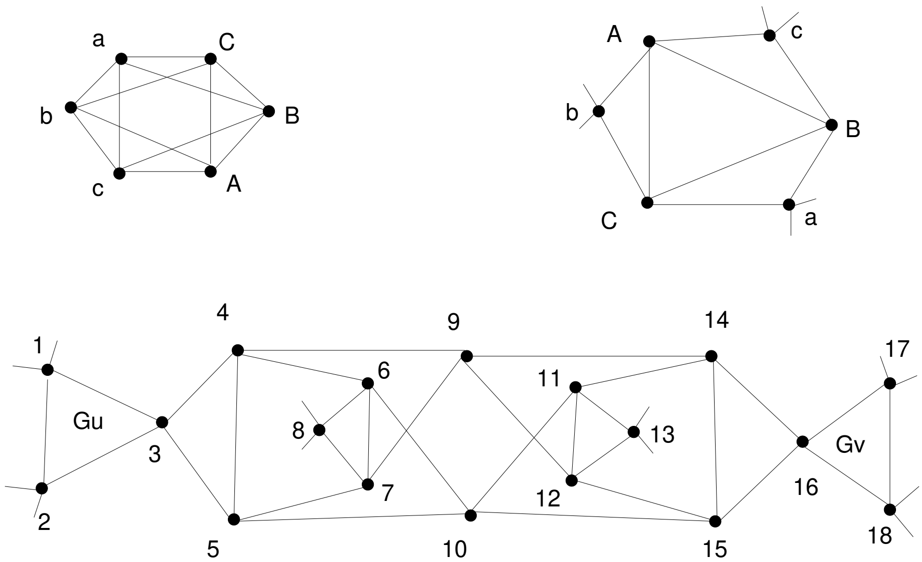


Figure 4.2: $K_{2,2,2}$ upper left. Vertex gadgets Gu and Gv connected by an edge gadget at bottom. Lacking neighbors for a, b, c provided by the 3-cycle A, B, C at top left.

3. Neighborhoods left unspecified are completed, possibly with added vertices, so that the 'if' direction of the reduction is met (this amounts to extending any partial covering projection defined in step 2 to a cover of H).

Theorem 4.6 The $K_{2,2,2}$-cover problem and the $\overline{C_6}$-cover problem are both $\mathcal{NP}$-complete.

Proof. Let $H = K_{2,2,2}$. For a given cubic graph G we construct a graph G' such that G is edge 3-colorable if and only if G' covers H. The gadget for a vertex $v \in V(G)$ is a 3-cycle, with one vertex for each port. The association of $V(H)$, as labels of the vertex gadget ports, with the 3 edge colors is that each pair of non-adjacent vertices of H corresponds to a unique color.

For the next step of this reduction, we define the edge gadget for $uv \in E(H)$ as a subgraph of a cover of H, see Figure 4.2. We show that if the graph we are constructing covers H by a projection f then the two ports connected to this edge gadget, called 3 and 16 in the figure, must be labelled by the same color. The vertex gadget cannot have two vertices labelled by two non-adjacent vertices of H. Assume wlog that $f(1) = b, f(2) = c, f(3) = a$, see Figure 4.2. We show that $f(16) = f(3) = a$. We have $N_H(a) = \{a, c, B, C\}$ so wlog let $f(4) = B, f(5) = C$. Since $N_H(C) = \{a, b, A, B\}$ we have $f(7)$ equal to A or b, and similarly $f(6)$ equal to A or c. But if $f(7) = A$ then $f(10) = b$ so $f(6) = c$ and $f(9) = A$ which cannot be since vertices 7 and 9 are adjacent. We conclude $f(7) = b, f(10) = A$ and similarly $f(6) = c, f(9) = A$. This forces $f(8) = a$. Now, $f(11)$ is b or B and similarly $f(12)$ is c or C. Any of the 4 possible pairs for $f(11), f(12)$ have as common neighbors only a

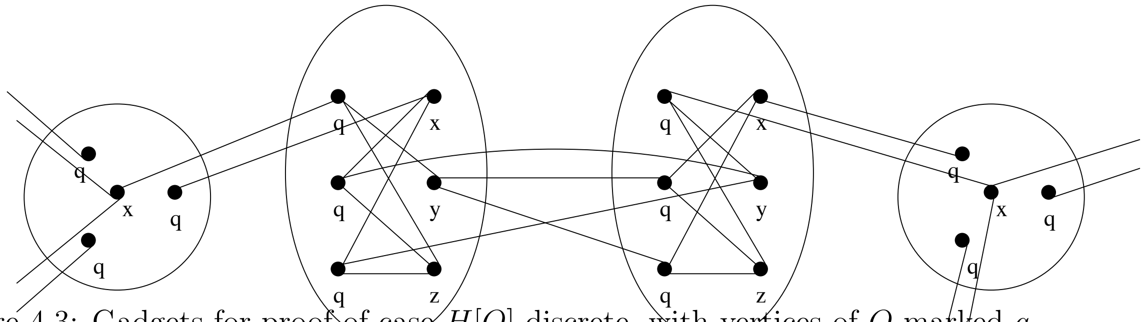


Figure 4.3: Gadgets for proof of case $H[Q]$ discrete, with vertices of Q marked q

and A, but 11 and 12 already have a neighbor labelled A, so $f(13) = a$. It is not hard to check that for all 4 cases we have $f(3) = f(16) = a$ so that G is edge 3-colorable whenever the constructed graph covers H.

Case 1: $f(11), f(12) = b, c \Rightarrow f(14), f(15) = C, B \Rightarrow f(16) = a$.
Case 2: $f(11), f(12) = B, c \Rightarrow f(14), f(15) = C, b \Rightarrow f(16) = a$.
Case 3: $f(11), f(12) = b, C \Rightarrow f(14), f(15) = c, B \Rightarrow f(16) = a$.
Case 4: $f(11), f(12) = B, C \Rightarrow f(14), f(15) = c, b \Rightarrow f(16) = a$.

For the other direction of the proof, we complete the neighborhoods left unspecified. Note that we can now assume that G is 3-edge colorable and freely specify the covering projection. For each $u \in V(G)$, we have three vertices, *e.g.* vertex 8 for u in Figure 4.2, each lacking two neighbors. Following the projection given above, let these three vertices be labelled a, b, c, lacking neighbors $\{B, C\}, \{A, C\}$ and $\{A, B\}$, respectively. We add a 3-cycle for each $u \in V(G)$, label its vertices A, B, C and use them as the lacking neighbors, see Figure 4.2. The constructed graph G' thus covers H whenever G is edge 3-colorable.

The reduction for $\overline{C_6}$, which we leave out, is again from edge 3-coloring of cubic graphs. □

We next give a series of $\mathcal{NP}$-completeness results for graphs whose vertices have a partition Q, R, S, with Q a block in the degree-partition whose vertices share all their neighbors S and induce a certain subgraph.

> **Theorem 4.7** The H-cover problem is $\mathcal{NP}$-complete if for some block Q in the degree-partition of H, $H[Q]$ is a discrete graph, $|Q| = 3$ and $\exists S \subseteq V(H), |S| \geq 3$ such that $\forall v \in Q, N(v) \setminus Q = S$.

Proof. We have $H[Q]$ a discrete graph on 3 vertices and $|S| \geq 3$. For a given cubic graph G, we construct a graph G' such that G' covers H if and only if G is 3-edge colorable. Let $\{x, y, z\} \subseteq S$ and $Q = \{q_1, q_2, q_3\}$. The vertex gadget V_v for a vertex $v \in V(G)$ will consist of an almost complete copy of H but lacking the edges connecting x and Q, xq_1, xq_2, xq_3. The edge gadget for an edge $uv \in E(G)$ will consist of two almost complete copies of H, call them E_{uv}, E_{vu}, each lacking the edges xq_1, yq_2, zq_3. The edges yq_2, zq_3 will instead connect together E_{uv} and E_{vu} by a total of four edges, ensuring that in a successfull cover of H we have both copies of q_2 (similarly both copies of q_3) in E_{uv} and E_{vu} labeled by the same vertex, which in turn will imply that both copies of q_1 have the same label. The edge gadget is connected to the vertex gadgets by an edge from x of V_v to q_1 of E_{vu} and an edge from x of E_{vu} to one of the Q-vertices of V_v, say q_{a_u} (the other Q-vertices of V_v are connected to the remaining two edge gadgets adjacent to V_v). Similar edges xq_1 and xq_{a_v} are added for E_{uv}, V_u, completing the construction of G'. See Figure 4.3. Note that in a cover of H by G' the two copies of q_1 in the gadget of edge uv and q_{a_u}, q_{a_v} in vertex gadgets of v and u, respectively, all receive the same label, corresponding to the unique color of edge (u, v).

Assuming G is 3-edge colorable with the edge uv being assigned the color $c \in \{1, 2, 3\}$ we label both copies of q_1 of the edge gadget of (u, v) by q_c. Since any permutation of $V(H)$ which moves only Q is an automorphism of H, this labeling is easily extended to make a cover of H by G'.

In the other direction, if G' covers H then since Q is a block in the degree partition of H, if we color edge uv of G by the label of both copies of q_1 in edge gadget of uv the result will be a 3-coloring of $E(G)$ such that edges incident with the same vertex receive distinct colors. $\square$

Theorem 4.8 The H-cover problem is $\mathcal{NP}$-complete if for some block Q in the degree-partition of H, $H[Q]$ is a k-cycle ($k \geq 3$) and $\exists S \subseteq V(H), |S| \geq 1$ such that $\forall v \in Q, N(v) \setminus Q = S$.

Proof. For a given graph G, we construct a graph G' such that G can be k-colored if and only if G' covers H. The gadget for a vertex v is a cycle Cv of length $deg_G(v) \times k$. In the case of a positive answer Cv will cover $H[Q]$. Cv is broken naturally into $deg_G(v)$ consecutive paths of length k, one for each edge incident with v, so that the first endpoint of each path receives the same label in any cover of $H[Q]$ by Cv. We call these endpoints the designated vertices of the gadget, with their label providing the corresponding color of vertex v. The gadget for an edge uv hooks up with one of these paths, say $c_1^u, ..., c_k^u$ from Cv and also with $c_1^v, ..., c_k^v$ from Cu. The edge gadget itself consists of the k-cycles $C1, ..., Ck-2$, cycle Ci having consecutive vertices $c_1^i, ..., c_k^i$, together with the $(H \setminus Q)$-copies $R_1, ..., R_k$. Vertices $c_i^v, c_i^u, c_i^1, c_i^2, ..., c_i^{k-2}$ are given all their remaining adjacencies (S-neighbors) from R_i, for $i = 1, ..., k$. This also satisfies R_i locally and completes the description of G', see Figure 4.4 for an example.

Assume $f : V(G)' \to V(H)$ is a covering projection and let $uv \in V(G)$ as above. Since Q is a block in the degree partition of H, the vertices of vertex gadgets must be sent to Q. The designated vertices c_1^u and c_1^v share a neighbor in R_1 ($|S| \geq 1$) and hence $f(c_1^u) \neq f(c_1^v)$. Hence we use the labels of the designated vertices of vertex gadgets as a k-coloring of the graph G.

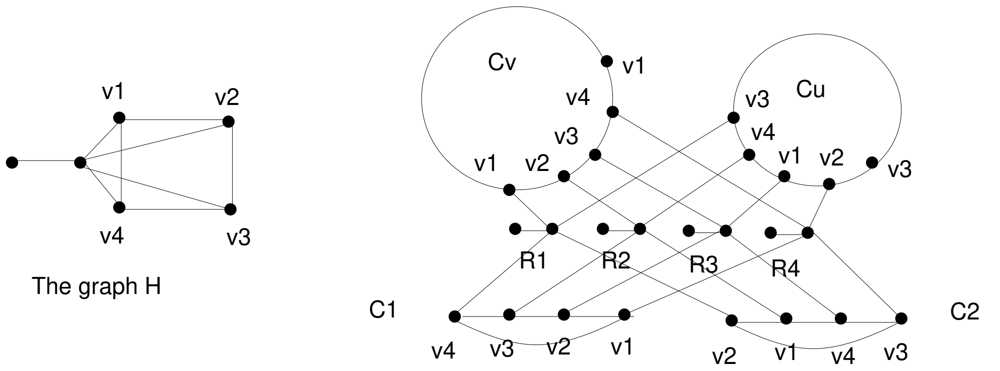


Figure 4.4: Case $H[Q]$ a 4-cycle. Vertex gadgets Cv and Cu and an edge gadget consisting of $R_1, R_2, R_3, R_4, C1, C2$, labeled so that vertices v and u are colored $v1$ and $v3$, respectively.

Assume G can be colored with colors $1, 2, ..., k$, with $Q = \{v_1, ..., v_k\}$. A vertex v colored c has its designated vertices labeled v_c in a cover of $H[Q]$ by its vertex gadget. In the gadget for an edge uv the $(H \setminus Q)$-copies naturally cover $H \setminus Q$. The k-cycles in the edge gadget are labeled uniquely to cover $H[Q]$ while satisfying neighborhoods of $(H \setminus Q)$-copies. This will result in a labeling where $c_i^v, c_i^u, c_i^1, c_i^2, ..., c_i^{k-2}$, with naming conventions as above, are given k distinct labels from Q, with G' covering H, see Figure 4.4 for an example. $\square$

Theorem 4.9 The H-cover problem is $\mathcal{NP}$-complete if for some block Q in the degree-partition of H, $\overline{H[Q]}$ is a k-cycle ($k \geq 5$) and $\exists S \subseteq V(H), |S| \geq 1$ such that $\forall v \in Q, N(v) \cap Q = S$.

Proof. For the case $\overline{H[Q]}$ a 5-cycle we have $H[Q]$ also a 5-cycle and use Theorem 4.8. For $k \geq 6$ and $\overline{H[Q]}$ a k-cycle we have $H[Q]$ connected and follow a proof very similar to that of the proof just given. The unique ordering when given a starting point (up to reflection) of adjacent vertices around the k-cycle used in the proof of Theorem 4.8 is replaced by the identical unique ordering of non-adjacent vertices around $H[Q]$, and an analogous reduction is used. For edge gadgets replace the $k-2$ k-cycles in the earlier proof by $k-2$ complements of k-cycles, $\overline{C_k}$. For vertex gadgets use the same number of vertices, visualized in a circle. No vertex will be adjacent to its 1st successor or 1st predecessor in the circle, but will be adjacent to its ith successor and ith predecessor in the circle, for $2 \leq i \leq \lfloor (k-1)/2 \rfloor$.

We consider two cases, k odd or k even, and argue that in a covering projection f of $\overline{C_k}$ by this vertex gadget, every kth vertex around the circle has the same label. For k odd, consider a vertex x and assume (wlog) $f(x) = v_1$. We show that the successor and predecessor of x are labeled by the two unique vertices of $\overline{C_k}$ which are not adjacent to v_1, namely v_k, v_2. Let the ith successor and predecessor of x be $s^i(x)$ and $p^i(x)$, respectively. We have $N(s(x)) - N(x) = \{p(x), s^{(k+1)/2}(x)\}$ hence

x and $s(x)$ share $k-5$ neighbors and $f(s(x))$ must be one of $\{v_1, v_2, v_3, v_{k-1}, v_k\}$. But $f(s(x)) = v_1$ means that $s^3(x)$ gets two v_1 neighbors. If $f(s(x)) = v_3$, since $v_1 \in N(v_3) - N(v_1)$ we must have either $f(p(x)) = v_1$ or $f(s^{(k+1)/2}(x)) = v_1$, but both possibilities mean that $f(s^2(x))$ gets two v_1 neighbors. Similarly, we cannot have $f(s(x)) = v_{k-1}$ and conclude that $f(s(x))$ is either v_k or v_2. The argument for $f(p(x))$ is analogous and since $p(x), s(x)$ are adjacent we conclude that the covering projection f enforces the same label for every kth vertex around the circle, as desired.

If k even, in addition to these connections, alternating sequences of $k/2$ vertices around the circle will be adjacent to their $k/2$th successors or $k/2$th predecessors, respectively. These sequences are thus naturally paired up in blocks of k vertices. In a covering projection f of $\overline{C_k}$ by this vertex gadget let x be the $k/2$th vertex of a given block B of k vertices and assume (wlog) $f(x) = v_1$. Since $|N(s(x)) \cap N(x)| = k - 4$ (they share every vertex in B but $\{p(x), x, s(x), s^2(x)\}$), we must have $f(s(x))$ be v_2 or v_k and assume (wlog) that $f(s(x)) = v_2$. Since $B = N(x) \cup N(s(x))$ we must have $f(B) = \{v_1, v_2, ..., v_k\}$. The only vertex in $B - N(x)$ which has not yet been assigned a label is $p(x)$ and the only vertex in $\{v_1, v_2, .., v_k\} - N(f(x))$ which is not already covered by a vertex of B is v_k, so $f(p(x)) = v_k$. Similarly, the only vertex in $B - N(s(x))$ which has not yet been assigned a label is $s^2(x)$ and the only vertex in $\{v_1, v_2, ..., v_k\} - N(f(s(x)))$ which is not already covered by a vertex of B is v_3, so $f(s^2(x)) = v_3$. Since $f(s(x)) = v_2$ and $f(s^2(x)) = v_3$ have already been assigned, if $f(s^3(x)) \neq v_4$ then $f(s^3(x)) \in N(v_3)$. But $s^2(x)$ is not adjacent to $s^3(x)$ and $N(s^2(x)) - \{B \cup N(s^3(x))\} = \emptyset$, so $s^2(x)$ could not get its $f(s^3(x))$ neighbor. We conclude that $f(s^3(x)) = v_4$. The above argument showing that $f(s^3(x)) = v_4$ so that the successor and predecessor of $s^2(x)$ are labeled by the two unique vertices of $\overline{C_k}$ to which $f(s^2(x))$ is not adjacent can be generalized to all but two of the remaining vertices of B if applied in the order $s^3(x), s^4(x), ..., s^{(k-2)/2}(x), p^2(x), p^3(x), ..., p^{(k-4)/2}(x)$. For the remaining two vertices $s^{k/2}(x)$ and $p^{(k-2)/2}(x)$ we thus have two choices, but the wrong choice for $s^{k/2}(x)$ entails that $s^2(x)$ would get two neighbors with the same label. In addition, if the vertex of G represented by this vertex gadget has degree larger than one, the vertex $s^2(x)$ has a single neighbor $s^{(k+2)/2}(x)$ in a neighboring block to B so that $f(s^{(k+2)/2}(x))$ is fixed to be $f(p^{(k-2)/2}(x))$ which in turn fixes all labels for that neighboring block. This suffices to enforce the same label for every kth vertex around the circle. With this observation in mind, the remainder of the proof follows the same logic as the case $H[Q]$ a k-cycle. $\square$

Theorem 4.10 The H-cover problem is $\mathcal{NP}$-complete if for some block Q ($|Q| = 2k \geq 4$) in the degree-partition of H, $H[Q]$ is a perfect matching and $\exists S \subseteq V(H), |S| \geq 2$ such that $\forall v \in Q, N(v) \setminus Q = S$.

Proof. We first consider the case $H[Q]$ a perfect matching on two edges. Let $Q = \{a_1, b_1, a_2, b_2\}$ with $\binom{Q}{2} \cap E(G) = \{a_1b_1, a_2b_2\}$. For a given graph G, we construct a graph G' such that G can be 4-colored if and only if G' covers H. Let $A \in S$. Let

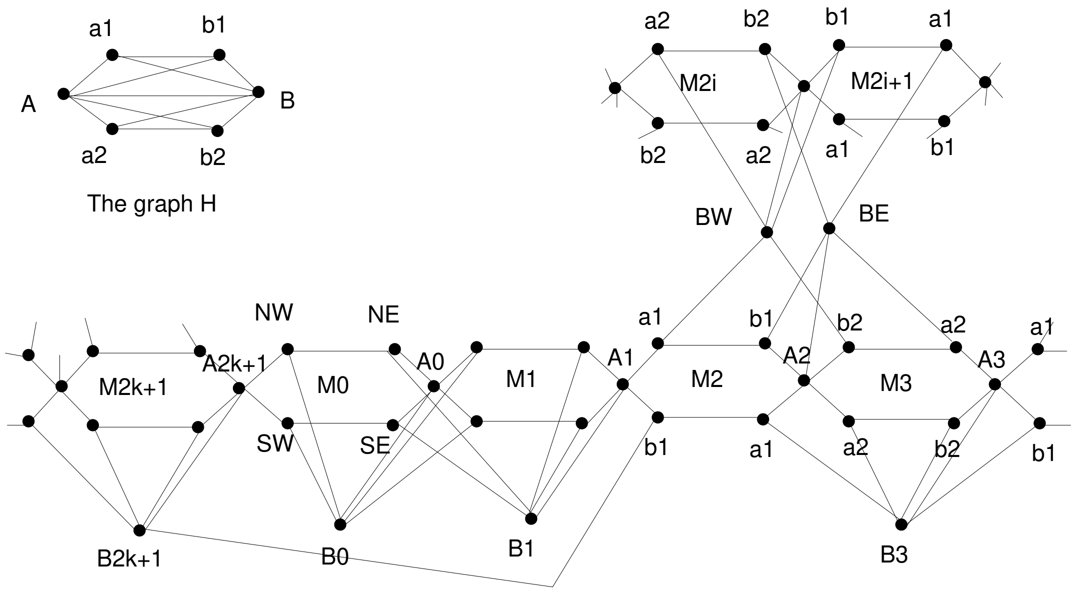


Figure 4.5: Case $H[Q]$ a matching. Vertex gadget for a vertex v of degree k and color a_1 at bottom, edge gadget vertices BW, BE, connected to a vertex colored a_2 at top. Expanding copies of Bx will give the general case $H[Q]$ a matching on 2 edges.

$v \in V(G)$ with $deg_G(v) = d$. The gadget v will consist of $2(d+1)$ copies of both $H[Q]$ and A, call these $M_0, ..., M_{2d+1}$ and $A_0, A_1, ..., A_{2d+1}$, respectively. Let the four vertices of M_i be NW_i, NE_i, SW_i, SE_i, with edges NW_iNE_i and SW_iSE_i. The copies of $H[Q]$ and A will form a cycle of 6-cycles by having the vertex A_i $\forall i$ connected to $NE_i, SE_i, NW_{i+1}, SW_{i+1}$ (addition modulo $2d+2$). The vertex gadget contains also $d+2$ copies of $H \setminus Q$, $L_0, L_1, L_3, L_5, ..., L_{2d+1}$, where L_i, for these subscripts, contains the previously described vertex A_i. Let S_i be the copy of $S = N(Q) \setminus Q$ in L_i. L_0 and L_1 play distinct roles and we describe their remaining connections first. Vertices $x \in S_0 \setminus A_0$ are connected to NW_0, SW_0, NW_1, SW_1, while vertices $x \in S_1 \setminus A_1$ are connected to NE_0, SE_0, NE_1, SE_1. Since Q is a block in the vertex partition of H, in a cover of H the edges of M_i $\forall i$ are sent to edges of $H[Q]$. Moreover, L_0 and L_1 enforce that to satisfy neighborhoods of A_0 and any $x \in S_0 \setminus A_0 \neq \emptyset$ all edges in M_{2i} $\forall i$ must be sent to a single edge of $H[Q]$, while edges of M_{2i+1} $\forall i$ are sent to the other edge of $H[Q]$. Vertices $x \in S_i \setminus A_i$ for $i = 3, 5, ..., 2d-1$ are connected to $SE_{i-1}, SW_i, SE_i, SW_{i+1}$, while any $x \in S_{2d+1} \setminus A_{2d+1}$ is connected to $SE_{2d}, SW_{2d+1}, SE_{2d+1}, SW_2$. This completes the description of the vertex gadget. Figure 4.5 gives the construction for a special case of H a 6-vertex graph, where L_i is the graph induced by B_i and A_i. In a cover of H, NW_{2i} and NW_{2i+2} must cover the same vertex, since otherwise a vertex $x \in S_{2i+1} \setminus A_{2i+1} \neq \emptyset$ will have two neighbors with the same label. Thus we have a unique label for NW_{2i} $\forall i$, which will correspond to the color of $v \in V(G)$ for the instance G of the 4-coloring problem. Note however, that the label of a NW_{2i+1} vertex is not fixed.

In this vertex gadget the only vertices lacking connections are A_{2i},NW_{2i}, NE_{2i}, NW_{2i+1},NE_{2i+1} for $i = 1, 2, ..., d$. The edge gadget for an edge uv consists of two copies of $H \setminus Q$, L_W and L_E, and will provide the remaining connections for some M_{2u_i}, M_{2u_i+1} from u's gadget and M_{2v_i}, M_{2v_i+1} from v's gadget. A_{2u_i} from u's gadget and A_{2v_i} from v's gadget are already contained in L_W and L_E, one in each. Additionaly, any $x \in S_W \setminus A_W$ is connected to the four NW vertices of these four copies of $H[Q]$ and any $x \in S_E \setminus A_E$ is connected to the four NE vertices. This ensures that labels of the NW vertex of M_{2u_i} and M_{2v_i} will differ while still allowing for any other combination from $\{a_1, b_1, a_2, b_2\}$.

The construction of G' is completed, see Figure 4.5. If G' covers H then we color G using the four colors $\{a_1, b_1, a_2, b_2\}$, with vertex v receiving the same color as the label of the NW_{2i} vertices in its gadget. By the observations made above, this constitutes a 4-coloring of G. Conversely, if G is vertex colorable by colors $\{a_1, b_1, a_2, b_2\}$, we label the NW_{2i} vertices of vertex gadgets accordingly. This labeling can be extended to a covering projection of H by G', see Figure 4.5 for an example where the adjacent vertices have colors a_1 and a_2.

We next consider the case when $H[Q]$ is a perfect matching of $2k$ edges, where $k \geq 3$. For a given graph G, we construct a graph G' such that G can be k-colored if and only if G' covers H. Let $A \in S$, and let $Q = \{a_1, b_1, ..., a_k, b_k\}$ with a_i, b_i adjacent in $H[Q]$. The gadget for a vertex v of G, will consist of $deg_v + 1$ copies of $H[Q]$, $M_0, M_1, ..., M_{deg_v}$ and $deg_v + 1$ copies of $H[V(H) - Q]$, $L_0, ..., L_{deg_v}$. In $V(L_i)$, let A_i be the copy of vertex A and S_i be the copies of S. Let A_i be adjacent to the copies of $a_j, \forall j$ in M_i and also to $b_j, \forall j$ in $M_{(i-1)\text{mod } deg_v}$, thus forming a $deg_v + 1$ 'cycle' of copies of $H[Q]$. M_0 will ensure that every copy of $H[Q]$ indeed maps to $H[Q]$ in a successfull cover, by having every vertex of $S_0 - A_0$ (not empty since $|S| \geq 2$) connected to every vertex of M_0. For $1 \leq i \leq deg_v$ connect every vertex of $S_i - A_i$ to every vertex of $V(M_i)$ except the copies of $\{a_1, b_1\}$. For $i > 1$ connect every vertex of $S_i - A_i$ to the copy of b_1 in $V(M_{i-1})$ and connect every vertex of $S_1 - A_1$ to the copy of b_1 in $V(M_{deg_v})$. This will enforce that in a covering projection of H, the copy of vertex a_1 in each of $M_1, ..., M_{deg_v}$ get the same label, determining the color of vertex $v \in V(G)$. The only vertices of a vertex gadget that have not yet been assigned all its neighbors are the copies of a_1 and $S_i - A_i$ in $M_1, ..., M_{deg_v}$.

The gadget for an edge (u, v) consists of an almost complete copy of H but leaving out the edges a_1, x and a_2, x for all $x \in \{S - A\}$. These remaining edges will be matched up with the missing edges for some M_{u_i} of u's gadget and M_{v_i} of v's gadget. Thus, a_1 (respectively a_2) of (u, v)'s gadget is made adjacent to every vertex in $S_{u_i} - Au_i$ of u's gadget (respectively $S_{v_i} - Av_i$ of v's gadget), and the copies of a_1 in both M_{u_i} of u's gadget and M_{v_i} of v's gadget are made adjacent to the copies of all $x \in \{S - A\}$ of (u, v)'s gadget. Thus, in a cover of H, assuming the copies of a_1 in u's gadget are labeled by a vertex a_i or b_i of Q, then in either case the copies of a_1 in v's gadget cannot be labeled a_i nor b_i since the copy of $H[Q]$ in (u, v)'s gadget must cover $H[Q]$, so that a vertex $x \in \{S - A\}$ would get too many a_i (or b_i) neighbors.

This completes the description of G'.

For sufficiency, suppose f is a covering projection of H by G'. Since Q is a block in the degree partition of H, copies of $H[Q]$ in G' must map to $H[Q]$. As we argued above, if a vertex $v \in V(G)$ is given the color i where in the vertex gadget of v the copies of a_1 map to a_i or b_i, then no two adjacent vertices in G receive the same color, and G is k-colorable. For the other direction, if vertex v of G receives color i in a k-coloring of that graph, we easily construct a covering projection of H by G' by labeling the copies of a_1 in v's gadget by a_i. $\square$

Theorem 4.11 The H-cover problem is $\mathcal{NP}$-complete if for some block Q ($|Q| = 2k \geq 4$) in the degree-partition of H, $\overline{H[Q]}$ is a perfect matching and $\exists S \subseteq V(H), |S| \geq 2$ such that $\forall v \in Q, N(v) \setminus Q = S$.

Proof. The case $\overline{H[Q]}$ a perfect matching of 2 edges implies $H[Q]$ a 4-cycle which is $\mathcal{NP}$-complete by Theorem 4.8. For the case that $\overline{H[Q]}$ is a perfect matching of $k \geq 3$ edges, we merely replace every copy of a matching of k edges in the construction given in the last proof by a copy of the complement of a matching of k edges. Since the only use we made of these copies was to pair up vertices by adjacencies, a pairing which can equally well be done by non-adjacencies, the exact same logic as given above shows that this latter case is also $\mathcal{NP}$-complete. $\square$

The *k-starfish* graph has k vertices of degree two and k vertices of degree four with the vertices of degree four inducing a cycle and any two consecutive vertices of this cycle sharing a neighbor of degree two, see Figure 4.6.

Theorem 4.12 For every $i \geq 1$ the $(2i + 1)$-starfish-cover problem is $\mathcal{NP}$-complete.

Proof. Let $k = 2i + 1$. Given a graph G, we construct a graph G' such that G is C_k-colorable if and only if G' covers the k-starfish. The vertex gadget Cv for $v \in V(G)$ consists of a cycle of length $k \times deg_G(v)$, broken naturally into $deg_G(v)$ consecutive paths of length k. The first endpoint of each path, the designated vertices of this gadget, receive the same label in any cover of a C_k-cycle. The edge gadget for $uv \in E(G)$ hooks up with two such paths, say $F_0 = c_1^0, c_2^0, ..., c_k^0$ from Cu and $F_1 = c_1^1, c_2^1, ..., c_k^1$ from Cv. The edge gadget contains the degree-2 vertices v_j^i for $i = 0, 2, ..., k-1$ and $j = 1, 2, ..., k$ and also contains $k-2$ k-cycles, call them $F_2, F_3, ..., F_{k-1}$, with F_i having consecutive vertices $c_1^i, c_2^i, ..., c_k^i$. $F_0, F_1, ..., F_{k-1}$ are hooked up by the degree-2 vertices to form a cycle, with c_j^i adjacent to v_j^i and to $v_j^{i+1 mod k}$. This completes the description of G', see Figure 4.6.

Let f be a cover of k-starfish by G', and let $uv \in E(G)$, with naming conventions as above. Since the designated vertices c_1^0 from Cu and c_1^1 from Cv have a common degree-2 neighbor, we must have $f(c_1^0)f(c_1^1)$ an edge in the k-starfish. Thus, we construct a C_k-coloring of G by focusing on the k-cycle induced by degree-4 vertices

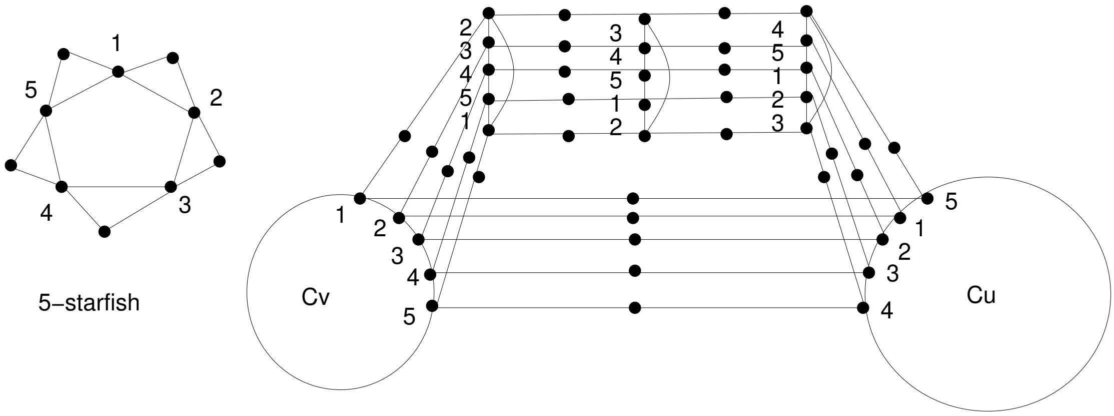


Figure 4.6: Case 5-starfish, with vertex gadgets Cu and Cv connected by an edge gadget.

in the k-starfish, and sending $u \in V(G)$ to the f-label of the designated vertex in its vertex gadget.

For the other direction of the proof we reverse this process, labeling designated vertices by the C_k-coloring induced on the degree-4 vertices of the k-starfish, see Figure 4.6 for an example. □

4.3.2 Reductions from Covering Problems

A graph H may have an induced subgraph H' for which the H'-cover problem is $\mathcal{NP}$-complete. In general, the H-cover problem could itself be easy. Our next theorem shows $\mathcal{NP}$-completeness in a restricted case by reducing the H-cover problem to the H'-cover problem.

> **Theorem 4.13** The H-cover problem is $\mathcal{NP}$-complete if for some block $Q = \{v_1, v_2, ..., v_k\}$ in the degree partition of H the $H[Q]$-cover problem is $\mathcal{NP}$-complete and there exists an order k latin square L over Q whose columns are elements of $Aut(H[Q])$, and whose rows are elements of $Aut(H \setminus E(H[Q]))|_Q$ (projections onto Q of automorphisms fixing Q setwise)

Proof. We reduce from the $H[Q]$-cover problem. Given a graph G, we construct a graph G' such that G covers $H[Q]$ if and only if G' covers H. Let $V(G) = \{x_1, ..., x_n\}$ and $V(H) = Q \cup R$. G' will contain k copies of G ($G_1, ..., G_k$) and n copies of $H[R]$ ($R_1, ..., R_n$). A vertex $x_i \in V(G)$ thus has k copies $x_i^1, x_i^2, ..., x_i^k$ in G' ($x_i^j \in V(G_j)$), which will be used as the remaining neighbors for vertices of R_i. We let the vertex x_i^j play the role of vertex $v_j \in Q$ and connect vertices of R_i to its remaining neighbors, as specified by H, thereby completing the construction of the graph G', see Figure 4.7 for an example.

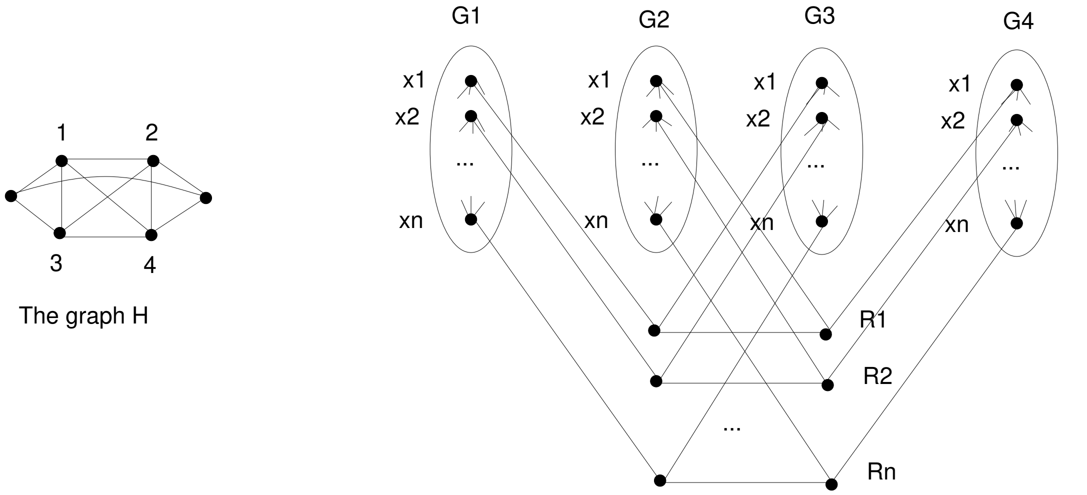


Figure 4.7: Case $H[Q] = K_4$, and the graph G' constructed by the reduction based on automorphism properties

Suppose G' covers H. Since Q is a block in the degree-partition of H the vertices of the n copies of $H[R]$ in G' cannot map to Q, so we have an n-fold cover. The vertices of each of the k copies of G must then map to Q and thus G covers $H[Q]$.

For the other direction, suppose $f : V(G) \to Q$ is a covering projection of $H[Q]$ by G. Let $\Delta_1, \Delta_2, ..., \Delta_k$ be the columns of the latin square L and let $\pi_1, \pi_2, ..., \pi_k$ be its rows. Since $\forall i : \Delta_i \in Aut(H[Q])$, we have by Fact 4.8 that $\Delta_1 \circ f, ..., \Delta_k \circ f$ are also cover projections of $H[Q]$ by G and we label the vertices of the copy G_j of G by $\Delta_j \circ f$. By construction we have that R_i is connected to vertices $x_i^1, x_i^2, ..., x_i^k$. Assuming that $f(x_i) = v_j$ we label these vertices by the respective labels $\Delta_1(v_j), \Delta_2(v_j), ..., \Delta_k(v_j)$, corresponding to a row π_r of L, when taken in this order. Since $\pi_r \in Aut(H \setminus E(H[Q]))|_Q$, we can send $V(R_i)$ to R by an element of $Aut(H \setminus E(H[Q]))$ which has projection π_r on Q, locally getting a covering projection from R_i to $H[R]$ by Fact 4.8, and with correct labels for remaining neighbors of R_i as well. The same is done for all n copies of $H[R]$ resulting in a mapping of $V(G')$ to $V(H)$ where each copy of G covers $H[Q]$ and each copy of $H[R]$ covers $H[R]$ and the remaining neighbors of copies of both G and $H[R]$ have correct labels, hence we have a covering projection of H by G'. $\square$

As an example of application of this result, consider the graph H depicted in Figure 4.7. $Q = \{v_1, v_2, v_3, v_4\}$ is a block in the degree partition of H inducing a complete graph and the K_4-cover problem is $\mathcal{NP}$-complete by Theorem 4.5. Moreover, for the following 4 by 4 matrix

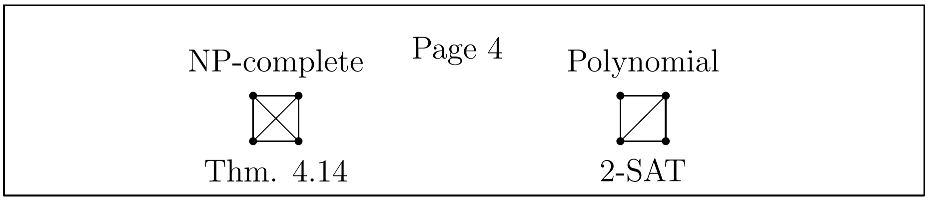
$$\begin{pmatrix} v_1 & v_2 & v_3 & v_4 \\ v_2 & v_1 & v_4 & v_3 \\ v_3 & v_4 & v_1 & v_2 \\ v_4 & v_3 & v_2 & v_1 \end{pmatrix}$$

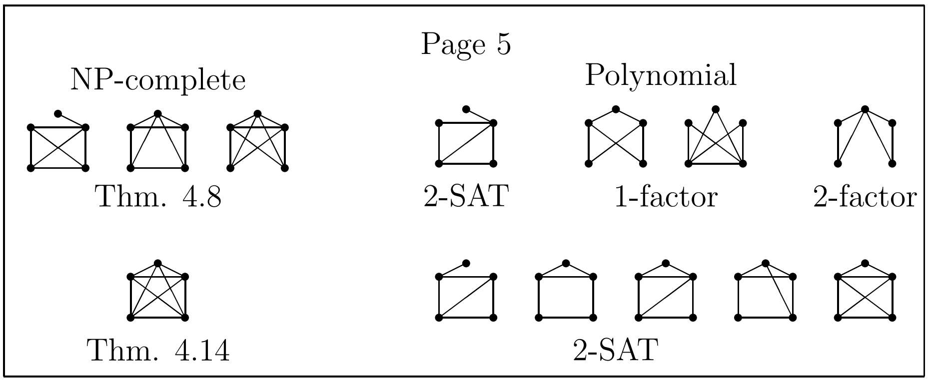
both its rows and columns are in $Aut(H)|_Q$, so by Theorem 4.13 the H-cover problem is $\mathcal{NP}$-complete (note $Aut(H)|_Q \subseteq (Aut(H \setminus E(H[Q]))|_Q \cap Aut(H[Q]))$.

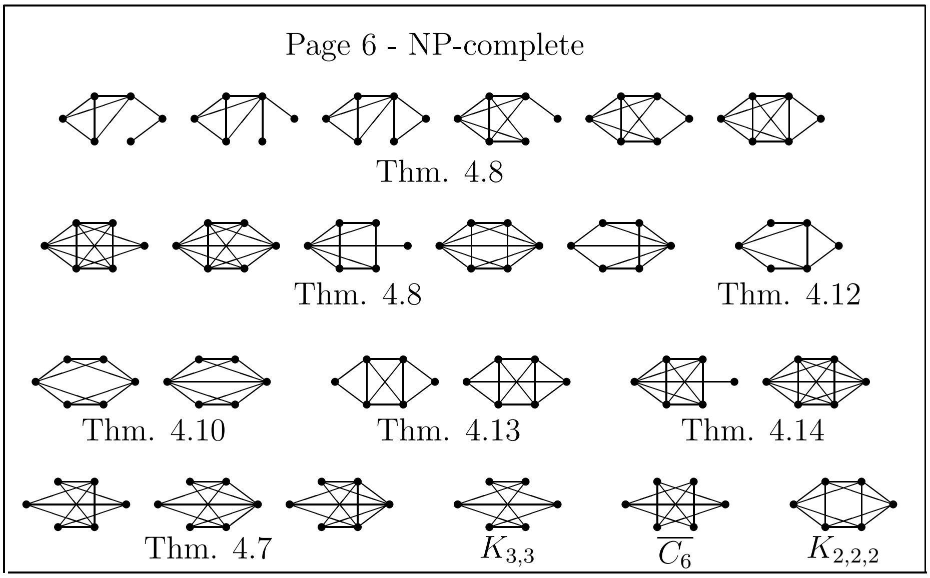
Theorem 4.14 The H-cover problem is $\mathcal{NP}$-complete if for some block Q ($|Q| \geq 4$) in the degree-partition of H, $H[Q]$ is a complete graph and $\exists S \subseteq V(H)$, such that $\forall v \in Q, N(v) \setminus Q = S$.

Proof. By Theorem 4.5 the $H[Q]$-cover problem is $\mathcal{NP}$-complete and $Aut(H)|_Q$ is the symmetric group on $|Q|$ points, so the conditions in Theorem 4.13 are easily satisfied. □

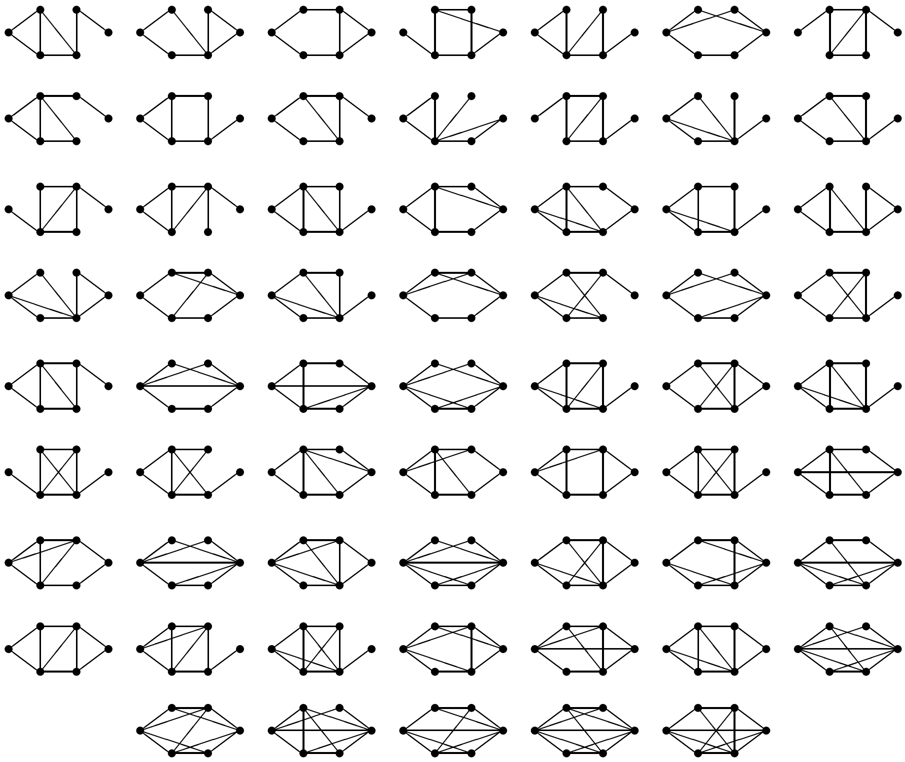
As an appendix, we list every connected, simple graph H on at most six vertices and at least two cycles, showing the complexity of the H-covering problem. Covering of simple graphs with at most one cycle is easy by Fact 4.7. By Facts 4.3 and 4.4 this resolves also the complexity of disconnected graphs having components on at most six vertices. The listing thus completes pages $1 \leq p \leq 6$ of the book on the complexity of the covering problem for simple graphs on p vertices.


Page 4
NP-complete
Polynomial
Thm. 4.14
2-SAT

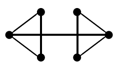

Page 5
NP-complete
Polynomial
Thm. 4.8
2-SAT
1-factor
2-factor
Thm. 4.14
2-SAT


Page 6 - NP-complete
Thm. 4.8
Thm. 4.8
Thm. 4.12
Thm. 4.10
Thm. 4.13
Thm. 4.14
Thm. 4.7
$K_{3,3}$
$\overline{C_6}$
$K_{2,2,2}$

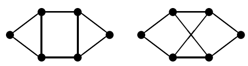
Page 6 - Polynomial



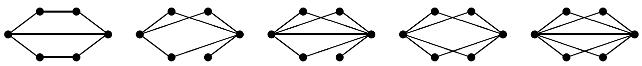
2-SAT



Thm. 4.4



Modified 2-SAT



1-factor

Chapter 5

Practical Partial k-Tree Algorithms

Many $\mathcal{NP}$-hard problems on graphs have polynomial, in fact usually linear as a function of input graph size, dynamic programming algorithms when restricted to partial k-trees (graphs of treewidth bounded by k), for fixed values of k. We improve on the practicality of such algorithms, both in terms of their complexity and their derivation, by accounting for dependency on the treewidth k. The algorithms are for fixed k; we consider a class of algorithms parameterized by k. We define a binary parse tree of partial k-trees which is based on two very simple graph operations. Then we discuss the derivation of dynamic programming solution algorithms, with a focus on the concepts of vertex states, separator states, table indices and equivalence classes of solutions. We also contrast our approach with related previous work.

5.1 Introduction

A graph G is a k-tree if it is a complete graph on k vertices or if it has a vertex $v \in V(G)$ whose neighbors induce a clique of size k and $G \setminus \{v\}$ is again a k-tree. The class of partial k-trees (the subgraphs of k-trees) is identical with the class of graphs of treewidth bounded by k. Many natural classes of graphs have bounded treewidth [53]. These classes are of algorithmic interest because many optimization problems, while inherently difficult (NP-hard) for general graphs are solvable in linear time on partial k-trees, for fixed k. These solution algorithms have two main steps, first finding a parse tree (tree-decomposition of width k [59]) of the input graph, and then computing the solution by a bottom-up traversal of the parse tree. For the first step, Bodlaender [17] gives a linear algorithm deciding if a graph is a partial k-tree and if so finding a tree-decomposition of width k, for fixed k. Unfortunately, the complexity of this algorithm as a function of the treewidth does not make it practical for larger values of k. For $k \leq 4$, however, practical algorithms based on graph rewriting do exist for the first step [10, 54, 60]. In this chapter we investigate the complexity of the second step as a function of k. There are many approaches to finding a template for the design of algorithms on partial k-trees with time complexity polynomial, or even linear, in the number of vertices [58, 7]. As a rule, proponents of these approaches have tried to encompass as wide a class of problems as possible, often at the expense of increased complexity as a function of k and also at the expense of simplicity of the resulting algorithms. Results giving explicit practical algorithms in this setting are usually confined to a few selected problems on either (full) k-trees [26], partial 1-trees or partial 2-trees [62, 36, 66]. We try to cover the middle ground between these two extremes, by investigating the time complexity as a function of both input size and the treewidth k. We define a binary parse tree of a partial k-tree which is easily derived from a tree-decomposition of the input graph. This parse tree is based on very simple graph operations, and we next show how this simplifies the design of dynamic programming algorithms on partial k-trees. We compare this approach to other strategies giving polytime algorithms on partial k-trees. Finally, we discuss a class of problems where a *vertex state* can be defined that will further ease the development and analysis of partial k-tree algorithms.

5.2 Binary Parse Tree

We give some standard definitions relating to partial k-trees before describing its binary parse tree. A partial k-tree G has at least k vertices and is a subgraph of a k-tree H, meaning $E(G) \subseteq E(H)$. The fact that we can assume $V(H) = V(G)$ follows from a simple technical lemma [6].

A k-tree H has a perfect elimination ordering of its n vertices, $peo = v_1, ..., v_n$ such that $\forall i : 1 \leq i < n-k$ the set of $k+1$ vertices $B_i = \{v_i\} \cup N_H(v_i) \cap \{v_i, ..., v_n\}$ induces a $(k+1)$-clique in H. The vertex v_i is *simplicial* in $H[\{v_i, v_{i+1}, ..., v_n\}]$, meaning that its closed neighborhood, B_i, induces a clique. It is not hard to show that $B_i \setminus \{v_i\}, i \in \{1, ..., n-k-1\}$ is a (minimal) separator of the graph H. See Figure 5.1 for an example of a partial 3-tree *embedded* in a 3-tree.

We call $B_i, 1 \leq i \leq n-k$ the $(k+1)$-*bag* of v_i in G under *peo* and each of its k-subsets is similarly called a k-bag of G under *peo*. The remaining definitions in this section are all for given $G, H, peo = v_1, ..., v_n$ and bags B_i as above. We first define a *peo-tree* P of G:

Definition 5.1 The peo-tree P of G based on *peo* has nodes $V(P) = \{B_1, ..., B_{n-k}\}$. The node B_i has as its parent in P the node B_j such that $j > i$ is the minimum bag index with $|B_i \cap B_j| = k$, except for the root B_{n-k} of P which has no parent.

The $(k+1)$-ary peo-tree P is a clique tree of H and also a width k tree-decomposition of both G and H. See Figure 5.2 for an example of a peo-tree.

We sketch an algebra on graphs with operations *Primitive*, *Reduce* and *Join*, needed to define a binary parse tree T of G based on the peo-tree P.

Definition 5.2 A *binary parse tree* of a graph G is the expression tree of an algebraic expression over the graph operations Primitive, Reduce and Join, evaluating to the graph G.

Let a graph with k distinguished vertices (also called sources, terminals, boundaries) have type (sort) G_k. For our purposes, sets of sources will always be $(k+1)$-bags and k-bags of G. We define the graph operations:

- *Primitive*: $\rightarrow G_{k+1}$. This 0-ary operation is used to introduce the graphs $G[B]$, for some $(k+1)$-bag B, as leaves of the parse tree.
- *Reduce*: $G_{k+1} \rightarrow G_k$. The unary operation Reduce takes the source vertex to be eliminated, which will be clear from context, and discards it as a source, leaving the graph itself unchanged.
- *Join*: $G_{k+1} \times G_k \rightarrow G_{k+1}$. The binary operation Join takes the union of its two argument graphs (A and B), where the sources of the second graph (a k-bag S_B) are a subset of the sources of the first graph (a $(k+1)$-bag S_A); these are the only shared vertices, and adjacencies for shared vertices are the same in both graphs. In other words, $V(A) \cap V(B) = S_B \subseteq S_A$ and $E(A[S_B]) = E(B[S_B])$, giving the resulting graph $Join(A, B) = (V(A) \cup V(B), E(A) \cup E(B))$ with sources S_A.

We employ these graph operations to describe the binary parse tree T of G based on the peo-tree P:

Definition 5.3 The binary parse tree T of G can be decomposed into $|V(P)|$ disjoint leaf-towards-root paths. Each of these paths is associated with a distinct node B_i of the peo-tree P. Let B_i have c children and parent $p(B_i)$. The path associated with B_i starts with the Primitive graph $G[B_i]$ as the leaf endpoint, has c Join operations as interior nodes and terminates with a node of a Reduce operation. The Reduce operation discards the vertex v_i as a source, and its node is the second child of the node of a Join operation associated with $p(B_i)$ (except for the root of T, which is the Reduce node in the path associated with the root node of P.)

Note the degree of freedom in the above definition in choosing parents for Reduce nodes. The parent of a Reduce node associated with B_i could be *any* one of the Join nodes associated with its parent $p(B_i)$. This degree of freedom, and also a possible choice of *peo*, can be exploited to keep the resulting parse tree shallow. We intend to investigate this possibility in future work on parallell partial k-tree algorithms. See Figure 5.3 for an example of a binary parse tree (note the leaf-towards-root paths are identified by starting at a Primitive node and moving towards the root until the first Reduce node is encountered, which forms the end of the path.)

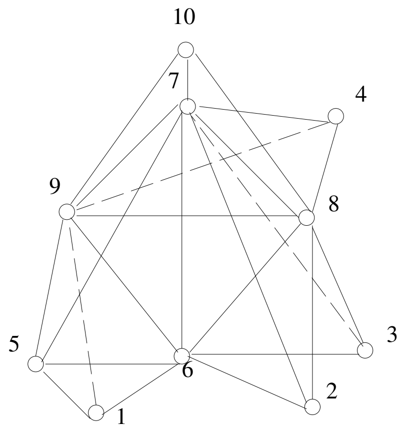


Figure 5.1: A partial 3-tree G, embedded in a 3-tree H, dashed edges in $E(H)-E(G)$, with peo=1,2,3,4,5,6,7,8,9,10

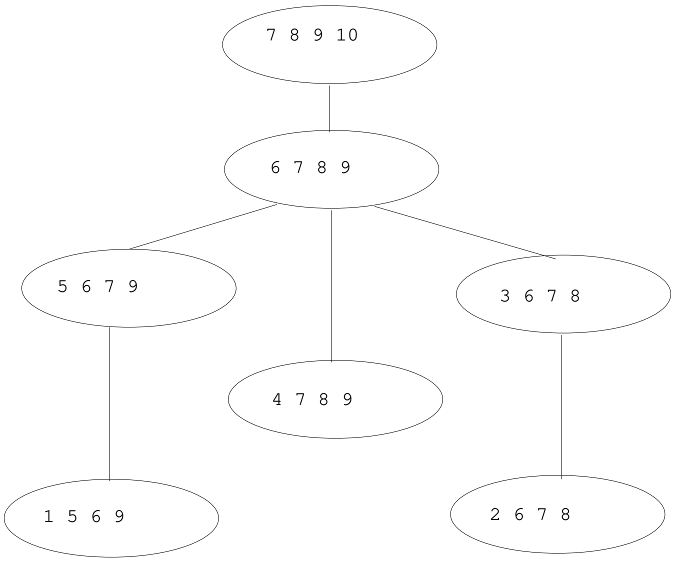


Figure 5.2: The peo-tree P of the partial 3-tree G.

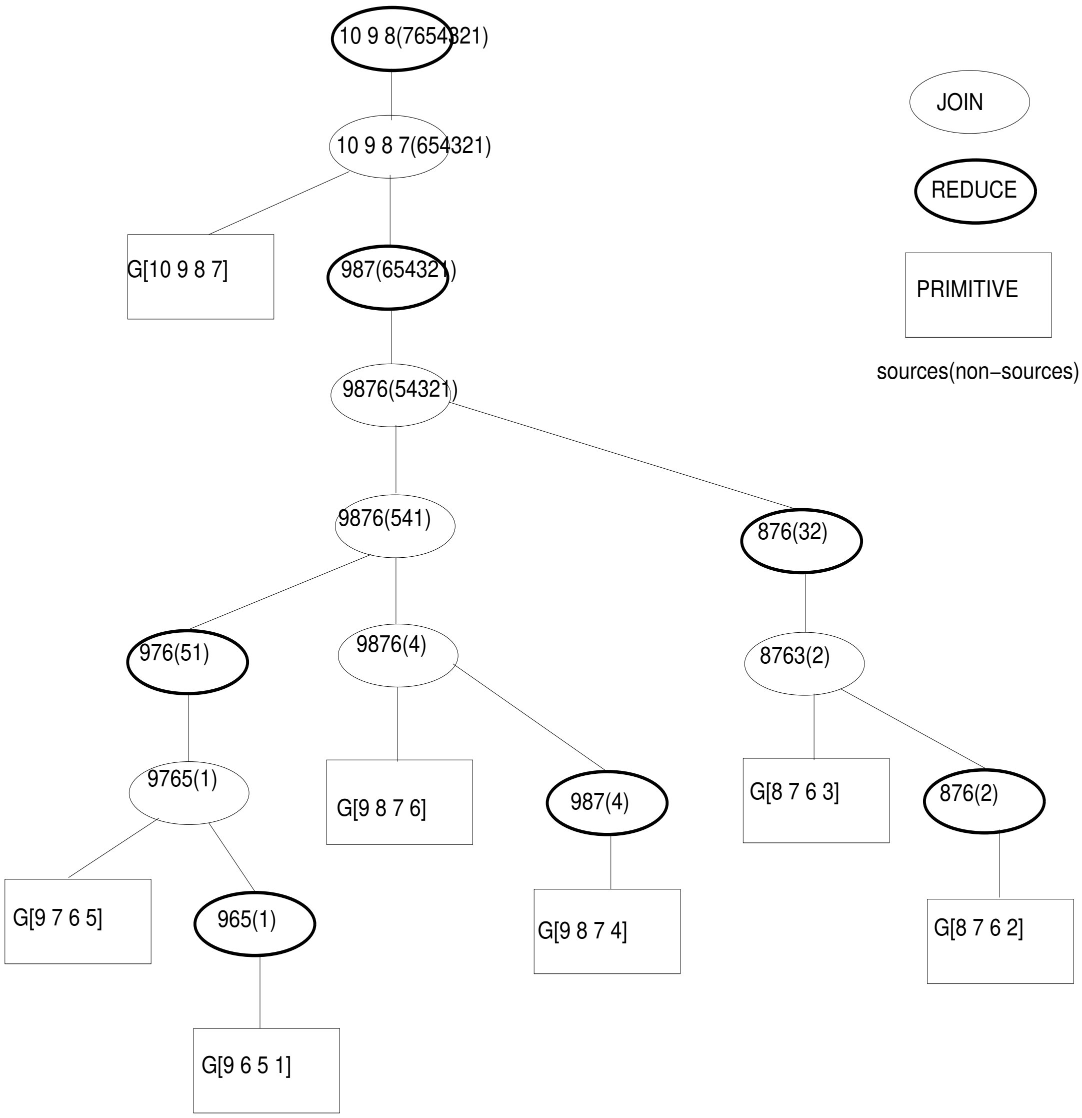


Figure 5.3: The binary parse tree T of the partial 3-tree G based on the peo-tree P. Nodes $u \in V(T)$ labeled by $V(G_u)$ with non-sources in parenthesis.

Note that the underlying algebraic expression for the binary parse tree T does indeed evaluate to G since the primitive graphs in T contain all vertices and edges of G, while Join and Reduce merely identify vertices of the primitive graphs, in the order given by P, to form G. We say that T *represents* G with sources $\{v_{n-k+1}, ..., v_n\}$. Since P is a tree with $n-k$ nodes, the Binary Parse Tree T of G has

- $n-k$ Primitive leaves, one for each node of P
- $n-k$ Reduce operations, one for each node of P
- $n-k-1$ Join operations, one for each edge of P

5.3 Dynamic Programming Algorithms

A dynamic programming solution algorithm for a problem R on G will follow a bottom-up traversal of the binary parse tree T. As usual, at each node u of T a data structure *table* is kept that contains optimal solutions to the problem R restricted to G_u, the subgraph of G represented by the subtree of T rooted at u. The table of a leaf is initialized according to the base case, the table of an interior node is filled in a bottom-up traversal of T based on tables of its children and the overall solution is obtained from the table at the root. The following information will complete the algorithm description for a given problem

- Description of Tables
- Operation Initialize-Primitive-Table
- Operation Join-Tables
- Operation Reduce-Table
- Operation Root-Optimization

An algorithm for a given problem must describe the tables involved and also describe how tables are updated. Derivation of algorithms starts with the definition of table indices. Each table index represents an equivalence class of solutions to subproblems, equivalent in terms of forming parts of larger solutions. A subproblem at a node u of the parse tree T is the problem R restricted to G_u and constrained on its sources. This subproblem solution interacts with the solutions to R on G only through the sources of G_u, which are a separator of G. Equivalent solutions affect the separator in the same manner, and hence we can define a *separator state* for each equivalence class (each table index). A candidate set of separator states is verified by the correctness proof of table update procedures for all operations involved. The introduction of the operations Reduce and Join greatly simplifies this verification process. In general, the algorithm computing a parameter $P(G)$ for a partial k-tree G given with a tree-decomposition follows the binary parse tree T of G, as follows:

Algorithm-R, where R is a graph parameter

Input: G, k,tree-decomposition of G of width k

Output: $R(G)$

(1) Find a binary parse tree T of G with Primitive, Reduce and Join nodes.

(2) Initialize Primitive Tables at leaves of T.

(3) Bottom-up traversal of T using Join-Tables and Reduce-Table.

(4) Table-Optimization at root of T gives $R(G)$.

For a given graph G on n vertices and any fixed k, Bodlaender [17] gives an $\mathrm{O}(n)$ algorithm (with a coefficient that is exponential in a polynomial in k) for deciding if the treewidth of G is at most k and in the affirmative case finding a tree-decomposition of G of width k. From this tree-decomposition it is straightforward to find a binary parse tree of G in time $\mathrm{O}(nk^2)$, see *e.g.* [53] for how to find an embedding in a k-tree, then find a *peo* and finally follow the description given in section 2 of this chapter for constructing the binary parse tree.

5.4 Comparisons with Related Work

Many strategies have been proposed for solving problems on graphs of bounded treewidth using a variant of the dynamic programming described above. We can classify these strategies by whether there is a procedure for automatic (mechanical) construction of a solution algorithm from a formal description of the problem, or such an algorithm has to be constructed "by hand".

One of the many automatic techniques, see *e.g.* [16, 28, 20], is the EMSOL approach of Arnborg, Lagergren and Seese [8], influenced by work of Courcelle [27]. A linear time algorithm solving a given problem can be constructed automatically from the problem description in the logic formalism *extended monadic second-order logic* (EMSOL). Although very powerful for showing assymptotic complexity results, this technique, and others like it, are unsatisfactory for practical algorithm design since the resulting complexity in k involves towers of powers of k. Currently, there is no automatic algorithm design strategy giving algorithms which are practical for increasing values of k without hand derivation of a large part of the algorithm.

The dynamic programming strategy on partial k-trees of Arnborg and Proskurowski [11], differs from our approach primarily in that a vertex v_i in a $(k+1)$-bag B was eliminated by combining tables of all $k+1$ k-bags in B in a single $(k+1)$-ary operation. Assuming the table for a k-bag has index set I_k, this operation has complexity $\Omega(|I_k|^{k+1})$ when all combinations of entries from each table are considered. Intuitively, the binary parse tree approach described above replaces a single such $(k+1)$-ary operation by at most k pairs of binary Join and Reduce operations, for complexity

$\mathcal{O}(k|I_k||I_{k+1}|)$. Moreover, the single $(k+1)$-ary operation used in the strategy of [11] is more complicated than the Join and Reduce operations employed here. Naturally, this plays an important role in the practical development of the algorithms.

We contrast our approach with dynamic programming algorithms that directly follow a tree-decomposition, as defined by Robertson and Seymour [59]. A *tree-decomposition* of width k of a graph G is a tree D where each node w of D is assigned a set $X_w \subseteq V(G)$ such that (i) $|X_w| \leq k+1$, (ii) if $uv \in E(G)$ then $\exists w \in V(D)$ with $\{u,v\} \subseteq X_w$ and (iii) for any $v \in V(G)$ the subgraph induced in D by the nodes $\{w : v \in X_w\}$ is connected. We have mentioned before that a graph is a partial k-tree iff it has a tree-decomposition of width k. Such a tree-decomposition is often used as the basis for a dynamic programming algorithm. Filling in the details of table updates in such algorithms is complicated by the generality of the tree-decomposition definition, which leads to a multitude of different cases of table operations. Although many algorithmic results follow a "nice" tree-decomposition, usually with the underlying tree being binary, we have not seen any previous approach which drastically lowers the number of different graph operations involved. The binary parse tree we defined above can itself be viewed as a nice tree-decomposition with particularly strong restrictions on the sets of vertices identified with each node in the binary tree. However, we believe it is most naturally described and understood in terms of the partial k-tree terminology.

There are several previous approaches to good algorithms on tree-like graphs that do employ a few simple graph operations [67, 14]. In particular, these strategies have been restricted to classes of graphs originally *defined* by an algebra on a few graph operations. A parse tree must be found in terms of these graph operations, and a dynamic programming strategy must be devised where table operations are designed according to the graph operations. Our approach can be viewed as providing a very simple algebra for constructing partial k-trees together with an algorithm for finding a parse tree founded on these operations from an arbitrary tree-decomposition. Certain approaches, *e.g.* Bern, Lawler and Wong [14], take an algebraic view of the resulting algorithms. For a class of graphs Γ given by an algebra over certain graph operations and primitive graphs, a subgraph property P is said to be *regular* if there is a homomorphism from $\Gamma_S = \{(G,S) : G \in \Gamma, S \subseteq V(G)\}$ to a finite set C_P with its own operations, which respects both the graph operations on Γ and the property P. In our case C_P corresponds to the set of equivalence classes of solutions making up the table index set for a particular problem, and the operations on C_P are the table update procedures given for each graph operation. A further comparison with this approach is given in Chapter 6.1.4 when we discuss algorithmic extensions to compute parameters defined over maximal and minimal vertex subsets.

5.5 Vertex State Problems

In section 3 we outlined the derivation of dynamic programming algorithms on partial k-trees using the method of bottom-up table updates along a parse tree of the input graph. Our approach to design such algorithms starts by defining the table indices involved. We look at a class of *vertex state* problems for which this process is particularly uniform. Each table index represents an equivalence class of solutions to subproblems, equivalent in terms of forming parts of larger solutions. A solution to a subproblem at a node u of the parse tree T is restricted to G_u. This subproblem solution interacts with the solutions to larger problems only through the sources of G_u, which constitute a separator of G. Equivalent solutions affect the separator in the same manner, and hence we can define a *separator state* for each equivalence class (table index).

For any vertex state problem, we can define a set A of *vertex states*, that represent the different ways that a solution to a subproblem can affect a single (source) vertex, such that $|A|$ is (usually) independent of n, and preferably also independent of k. For the Grundy Number problem, discussed in chapter 6.2.4, we will see that we can achieve a polytime algorithm even when $|A|$ is not independent of n. The Cartesian product of vertex states of separator vertices defines the state of the separator. A separator with k vertices has then a distinct separator state for each different k-vector of vertex states so its table index set has size $|I_k| = |A|^k$. For a vertex state problem R having a dynamic programming algorithm on partial k-trees as outlined in section 6.3, consider a node u, with sources B_u, of the binary parse tree T of an input graph G. The following sets are all in a natural one-to-one correspondence:

1. The equivalence classes of solutions to subproblems on G_u
2. The table index set for the node u of T
3. The set of separator states for the sources B_u of node u of T
4. The set of $|B_u|$-vectors of vertex states

We will not define vertex state problems explicitly, except to note that a decision problem asking for a partition of vertices for which a purported solution S can be verified by a simple local check of how the neighbors of each vertex intersects with S will be a vertex state problem. For instance, any of the vertex partitioning problems defined in Chapter 2.1 is a vertex state problem. On the other hand, the Hamilton cycle problem does not satisfy these criteria. One of the goals of our future research is to explore vertex state problems in a wider class than vertex partitioning problems.

To design algorithms for vertex state problems on partial k-trees we focus on the solution verification method through a local check of vertex neighborhoods. We first define a set of *legal* vertex states that will provide the information necessary to infer a solution to the original problem. In the case of vertex partitioning problems, a row of the degree constraint matrix is the starting-point for defining a set of legal vertex states. Note that for a subgraph G_u the vertex state information will be maintained only for the source vertices B_u. The subgraph G_u interacts with the remainder of the input graph G only through these sources. Vertex states must carry information telling us whether a suggested solution not satisfying the problem constraints on G_u can be augmented to good solutions on a supergraph of G_u by adding neighbors to its sources B_u. Such a suggested solution carries the need for additional information beyond that captured by the legal states. With this in mind, we define an equivalence relation on solutions to subproblems:

> **Definition 5.4** Two potential solutions S_1 and S_2 to a subproblem on G_u are equivalent if for every extension of S_1 and S_2 to S_1' and S_2', respectively, such that the new vertices are added as neighbors of B_u-vertices, with the new vertices playing the same role in S_1' and S_2', has the effect that either both S_1' and S_2' are solutions or none of them are.

This equivalence relation is a refinement of the original classification of solutions into simple yes/no classes. The set of vertex states needed to capture this refinement constitute the augmented vertex states, a superset of the legal vertex states. The set of vertex states A for a specific problem as mentioned earlier refers to the set of augmented vertex states as described here. A candidate set of augmented vertex states gives rise to a set of separator states. Finally, a candidate set of separator states is verified by the correctness proof of table update procedures for all operations involved. The introduction of the operations Reduce and Join greatly simplifies this verification process. See the next chapter for an example.

Let R be a vertex state problem with vertex state set A. In the next chapter, we will see that in our algorithm for solving R on a partial k-tree of n vertices, the most expensive operation is computation of the binary Join operation. The complexity of the Join operation at a node of the parse tree is proportional to the number of pairs of indices, one index from the table of each of its two children. The table index set for the problem R at a node with k sources has size $|A|^k$, and there are less than n Join nodes in the parse tree. The overall complexity of the algorithm, given a tree-decomposition, is then $T(n, k, L) = \mathcal{O}(n|L|^{2k+1})$ since the children of a Join node have k and $k+1$ sources, respectively. In the next chapter we show an application of these ideas.

Chapter 6

Algorithms for Vertex Partitioning Problems on Partial k-Trees

In this chapter we first give algorithms for solving vertex subset optimization problems on partial k-trees, based on the algorithm design template given Chapter 5 and the characterization of problems given in Chapter 2. We then extend these algorithms to the more general case of vertex partitioning problems. These algorithms take a graph G and a width k tree-decomposition of G as input. Earlier work by Arnborg et al. [8] establishes the existence of pseudo-efficient algorithms for most, but not all, of these problems. They are pseudo-efficient in the sense that their time complexity is polynomial in the size of the input for fixed k, but with horrendous multiplicative constants ("towers" of powers of k). In contrast to this behavior, the algorithms presented here have running times with more reasonable bounds as a function of both input size and treewidth, *e.g.* $\mathcal{O}(n2^{4k})$ for well-known vertex subset optimization problems. Since these problems are $\mathcal{NP}$-hard in general and a tree-decomposition of width $n-1$ is easily found for any graph on n vertices, we should not expect polynomial dependence on k. As an extension of our methodology we provide the first polynomial-time algorithms on partial k-trees for the Grundy Number, a problem not known to be expressible in EMSOL even when restricted to graphs of bounded treewidth [50], and neither known to have a finite-state description. This follows from (i) the description of the Grundy Number problem as a vertex partitioning problem, (ii) a new logarithmic bound on the Grundy Number of a partial k-tree, and (iii) the careful investigation of time complexity of vertex partitioning problems on partial k-trees.

6.1 Vertex Subset Algorithms

In general, the algorithm computing the vertex subset optimization problem $optM[L](G)$ for a partial k-tree G follows the binary parse tree T of G as outlined in the previous chapter. Recall that opt can be either max or min, with the $maxM[L]$ problem maximizing over all $[L]$-sets S in G the value of $|\{v : state_S(v) \in M\}|$, see Chapter

2.2. The somewhat easier algorithms for $\exists[L]$ problems can be seen as a special case of an optimization problem where $M = \emptyset$.

Algorithm-optM[L], where opt is either max or min.

Input: G, k,tree-decomposition of G of width k

Output: $optM[L](G)$

(1) Find a binary parse tree T of G with Primitive, Reduce and Join nodes.

(2) Initialize Primitive Tables at leaves of T.

(3) Traverse T in the bottom-up manner using Join-Tables and Reduce-Table.

(4) Table-Optimization at root of T gives $optM[L](G)$.

We first discuss the pertinent vertex and separator states and give a description of the tables involved in the algorithm. We then fill in details of table operations, prove their correctness and give their time complexities. Finally, we look at some extensions of this approach.

6.1.1 Vertex States

Before giving an algorithm to compute a parameter $optM[L]$, we discuss some issues related to the concept of vertex states in the algorithmic context. In Chapter 2, the notation $\sigma_{\geq 0}$ was used merely as an abbreviation for the infinite set of vertex states $\sigma_0, \sigma_1, ...$. The tables involved in the algorithms of this chapter will be indexed by all possible states that vertices can have, so for complexity reasons we want as few distinct vertex states as possible, and for certain problems, we will view, *e.g.*, $\sigma_{\geq 0}$ as a single *augmented* vertex state. For an $optM[L]$ problem, as defined in Chapter 2, we now define the set of *augmented* vertex states A and define $Astate_S(v)$ for $v \in V(G)$ and $S \subseteq V(G)$. For instance, consider dominating sets where $L = \{\rho_1, \rho_2, ...\} \cup \{\sigma_0, \sigma_1, ...\} = \{\rho_{\geq 1}, \sigma_{\geq 0}\}$. In our algorithms, a vertex may start out in the non-legal state ρ_0 and acquire σ-neighbors as the algorithm progresses, so we must allow vertices to have state ρ_0 at some point during the algorithm. On the other hand, for the minimum dominating set problem the algorithm need not discriminate ρ_1 from ρ_2 since vertices will not lose (σ-) neighbors in the course of the algorithm. For the same reason, an algorithm to find maximum independent sets, $max[\rho_{\geq 0}, \sigma_0]$ would not keep track of any σ-state other than σ_0. We must also ensure that we can optimize correctly, so for the problem $min\{\sigma_1\}[\rho_{\geq 1}, \sigma_{\geq 0}]$, we *will* need to discriminate between σ_1 and $\sigma_{\geq 2}$, getting the *augmented* vertex state set $A = \{\rho_0, \rho_{\geq 1}, \sigma_0, \sigma_1, \sigma_{\geq 2}\}$. We next define this formally:

Definition 6.1

For a problem $optM[L]$, define

$$M_\rho = \{i \in \mathbb{N} : \rho_\beth \in \mathbb{M}\} \qquad D_\rho = \{i \in \mathbb{N} : \rho_\beth \in \mathbb{L}\}$$
$$M_\sigma = \{i \in \mathbb{N} : \sigma_\beth \in \mathbb{M}\} \qquad D_\sigma = \{i \in \mathbb{N} : \sigma_\beth \in \mathbb{L}\}$$

For the above example, we would get $M_\sigma = \{1\}$ and $D_\rho = \{1, 2, 3, ...\}$. To formally define the range of subscripts for the augmented ρ-states, we need

Definition 6.2
$Y_t \stackrel{df}{=} \{0, 1, 2, ..., t\}$
$W_t \stackrel{df}{=} Y_{t-1} \cup \{\geq t\}$
$R \stackrel{df}{=} \{Y_t : t \in \mathbb{N}\} \cup \{\mathbb{W}_\approx : \approx \in \mathbb{N}\} \cup \{\mathbb{N}\}$

Note that $|Y_t| = |W_t| = t + 1$, *i.e.*, $\geq t$ is a single element of W_t. We now define a function $\alpha : 2^{\mathbb{N}} \times 2^{\mathbb{N}} \to R$ such that $\alpha(M_\rho, D_\rho)$ and $\alpha(M_\sigma, D_\sigma)$ gives the set of subscripts used in the algorithm for the augmented ρ-states and σ-states, respectively. We assume that $M_\rho \subseteq D_\rho$ and $M_\sigma \subseteq D_\sigma$ since optimizing over vertices with non-legal states is not interesting.

Definition 6.3

$$\alpha(M_\rho, D_\rho) = \begin{cases} Y_t & \text{if } \exists t \in D_\rho \text{ s.t. } t = \max\{D_\rho\} \\ W_t & \text{if } \exists t \in D_\rho \text{ with } t \text{ minimum s.t. } t > \max\{M_\rho\} \text{ and } \{t, t+1, ...\} \subseteq D_\rho \\ W_t & \text{if } \exists t \in D_\rho \text{ with } t \text{ minimum s.t. } \{t, t+1, ...\} \subseteq D_\rho, M_\rho \\ \mathbb{N} & \text{otherwise} \end{cases}$$

The definition for $\alpha(M_\sigma, D_\sigma)$ is analogous. We assume $0 > \max\{\emptyset\}$, and note that our algorithmic template will not capture the last case above when α returns $\mathbb{N}$, as this would imply an infinite augmented vertex state set. For our example we get $\alpha(M_\rho, D_\rho) = \alpha(\emptyset, \{1, 2, 3, ...\}) = W_1 = \{0, \geq 1\}$. We define formally the set of augmented states.

Definition 6.4 A problem *optM*$[L]$ has the augmented vertex state set
$A = \{\rho_x : x \in \alpha(M_\rho, D_\rho)\} \cup \{\sigma_x : x \in \alpha(M_\sigma, D_\sigma)\}$

For instance, the augmented vertex state set for $min\{\sigma_1\}[\rho_{\geq 1}, \sigma_{\geq 0}]$ is $A = \{\rho_x : x \in \alpha(\emptyset, \{1, 2, 3, ...\})\} \cup \{\sigma_x : x \in \alpha(\{1\}, \{0, 1, ...\})\} = \{\rho_x : x \in \{0, \geq 1\}\} \cup \{\sigma_x : x \in \{0, 1, \geq 2\}\} = \{\rho_0, \rho_{\geq 1}, \sigma_0, \sigma_1, \sigma_{\geq 2}\}$.

A central operation in our algorithms is $a \oplus b \ominus c$ which adds the subscripts of either two augmented ρ-states a, b or two augmented σ-states a, b and subtracts $c \in \mathbb{N}$. This operation returns the subscript of an augmented ρ-state or σ-state, respectively, unless undefined. The definition of $a \oplus b \ominus c$ depends on whether subscripts of augmented states a, b are over Y_t or W_t.

Definition 6.5 For $a, b \in Y_t$ and $c \in \mathbb{N}$

$$a \oplus b \ominus c = \begin{cases} a+b-c & \text{if } a+b-c \in Y_t \\ \uparrow & \text{otherwise} \end{cases}$$

For $a, b \in W_t$ and $c \in \mathbb{N}$

$$a \oplus b \ominus c = \begin{cases} \geq t & \text{if either } a \text{ or } b \text{ is } \geq t \\ \geq t & \text{if } a+b-c \in \{t, t+1, ...\} \\ a+b-c & \text{if } a+b-c \in \{0, 1, ..., t-1\} \\ \uparrow & \text{otherwise} \end{cases}$$

The notions defined above will also be used for solving vertex partitioning problems.

6.1.2 Table Description

We next describe the tables involved in an algorithm for a vertex subset optimization problem $optM[L]$ given by the legal states L, augmented states A and optimizing (min or max) the set of vertices with state in M. When using the notation $Astate_S(v)$ in the following, we follow the definition of augmented states. For instance, the algorithm for $min\{\sigma_{\geq 0}\}[\rho_{\geq 1}, \sigma_{\geq 0}]$, the Minimum Dominating Set problem, which has $A = \{\rho_0, \rho_{\geq 1}, \sigma_{\geq 0}\}$ uses the following natural interpretation of $Astate_S$ (for $S \subseteq V(G)$)

$$Astate_S(v) = \begin{cases} \rho_0 & \text{if } v \notin S \text{ and } |N_G(v) \cap S| = 0 \\ \rho_{\geq 1} & \text{if } v \notin S \text{ and } |N_G(v) \cap S| \geq 1 \\ \sigma_{\geq 0} & \text{if } v \in S \end{cases}$$

In this algorithmic context, we naturally view the legal states L and optimized vertex states M as a subset of the augmented states A. For instance, in the $min\{\sigma_1\}[\rho_{\geq 1}, \sigma_{\geq 0}]$ problem we have $L = \{\rho_1, \rho_2, ...\} \cup \{\sigma_0, \sigma_1, ...\}$, $A = \{\rho_0, \rho_{\geq 1}, \sigma_0, \sigma_1, \sigma_{\geq 2}\}$, and interpret the legal vertex state set L as $\{\rho_{\geq 1}, \sigma_0, \sigma_1, \sigma_{\geq 2}\} \subseteq A$. Let a node u of the parse tree T represent the subgraph G_u of G with sources $B_u = \{w_1, ..., w_k\}$. The table at node u, $Table_u$, has index set $I_k = \{\mathbf{s} = s_1, ...s_k : s_i \in A\}$, so that $|I_k| = |A|^k$. We define Ψ, with respect to G_u and $\mathbf{s}$, to be the family of sets $S \subseteq V(G_u)$ such that in the graph G_u, for $w_i \in B_u$, $Astate_S(w_i) = s_i, 1 \leq i \leq k$ and for $v \in V(G_u) \setminus B_u, Astate_S(v) \in L$.

Definition 6.6 For problem $optM[L]$ with augmented states A, graph G_u with sources $B_u = \{w_1, w_2, ..., w_k\}$ and k-vector $\mathbf{s} = s_1, ...s_k : s_i \in A$ we define

$\Psi \stackrel{df}{=} \{S \subseteq V(G_u) : \forall v \in V(G_u) \setminus B_u \ \forall w_i \in B_u \quad Astate_S(v) \in L$ and $Astate_S(w_i) = s_i\}$

Ψ forms an equivalence class of solutions to the subproblem on G_u, and its elements are called Ψ-sets respecting G_u and $\mathbf{s}$. Note that in a Ψ-set vertices of B_u are allowed to have any state from the augmented set A, whereas vertices already "eliminated" must have a legal state in L. The value of $Table_u[\mathbf{s}]$ is the optimum (max or min) number of vertices in $V(G_u) \setminus B_u$ that have state in M over all Ψ-sets respecting G_u and $\mathbf{s}$, and $\perp$ if no such Ψ-set exists.

Definition 6.7

$$Table_u[\mathbf{s}] \stackrel{df}{=} \begin{cases} \perp & \text{if } \Psi = \emptyset \\ optimum_{S\in\Psi}\{|\{v \in V(G_u) \setminus B_u : Astate_S(v) \in M\}|\} & \text{otherwise} \end{cases}$$

The result of an addition when one or more of the operands have the value $\perp$ is again $\perp$, and this value is considered to be smaller, respectively larger, than any integer under maximization, respectively minimization.

6.1.3 Table Operations

We now elaborate on the operations of Table-Initialization, Table-Reduce, Join-Tables and Table-Root-Optimization. Each of the following subsections defines the appropriate procedure, gives the proof of its correctness and analyzes its complexity.

Table Initialization. A leaf u of T is a Primitive node and G_u is the graph $G[B_u]$, where $B_u = \{w_1, ..., w_{k+1}\}$. Following the above definition we initialize $Table_u$ in two steps,

$$(1)\ \forall \mathbf{s} \in I_{k+1} : Table_u[\mathbf{s}] := \perp$$

$$(2)\ \forall S \subseteq B_u\text{: } Table_u[\mathbf{s}] := \begin{cases} 0 & \text{if } \mathbf{s} = s_1, ..., s_{k+1} \text{ and } Astate_S(w_i) = s_i \in A \\ & \qquad\qquad \text{in the graph } G[B_u] \\ \perp & \text{otherwise} \end{cases}$$

The complexity of this initialization for each leaf of T is $\mathcal{O}(|A|^{k+1} + 2^{k+2\log k})$.

Reduce Table. A Reduce node u of T has a single child a such that $B_u = \{w_1, ..., w_k\}$ and $B_a = \{w_1, ..., w_{k+1}\}$. We compute $Table_u$ based on correct $Table_a$ as follows

$$\forall \mathbf{s} \in I_k : Table_u[\mathbf{s}] := optimum\{Table_a[\mathbf{p}](+1 \text{ if } p_{k+1} \in M)\}$$

where the optimum (min or max) is taken over all $\mathbf{p} \in I_{k+1}$ such that $p_{k+1} \in L$ and $1 \leq i \leq k, p_i = s_i$. Correctness of the operation follows by noting that G_a and G_u designate the same subgraph of G, and differ only by w_{k+1} not being a source in G_u. By definition, an entry of $Table_u$ optimizes over solutions where the state of w_{k+1} is one of the legal states L and w_{k+1} contributes to the entry value if it has state in M. The complexity of this operation for each Reduce node of T is $\mathcal{O}(|A|^{k+1})$.

Join Tables. A Join node u of T has children a and b such that $B_u = B_a = \{w_1, ..., w_{k+1}\}$ and $B_b = \{w_1, ..., w_k\}$ is a k-subset of B_a. Moreover, G_a and G_b share exactly the subgraph induced by B_b, $G[B_b]$. We compute $Table_u$ by considering all pairs of table entries of the form $Table_a[\mathbf{p}], Table_b[\mathbf{r}]$. Recall that the separator state $\mathbf{p}$ consists of $k+1$ vertex states $p_1, p_2, ..., p_{k+1}$ where the state $p_i \in A$ is associated with vertex w_i. In the procedure for the Join operation, we first check that $\mathbf{p}, \mathbf{r}$ is a *compatible* separator state pair, meaning that for each $w_i, i \in \{1, ..., k\}$ both $\mathbf{p}$ and $\mathbf{r}$ agree on whether $w_i \in S$. To define this operation, let *selected*(*state*) for a vertex state be σ if *state* is a σ-state and ρ otherwise.

$$compatible(\mathbf{p}, \mathbf{r}) := \begin{cases} 1 & \text{if } selected(p_i) = selected(r_i) \ \forall i \in \{1, ..., k\} \\ 0 & \text{otherwise} \end{cases}$$

We then combine, for each $w_i, i \in \{1, ..., k+1\}$ the contributions from $\mathbf{p}$ and $\mathbf{r}$ to give the resulting separator state $combine(\mathbf{p}, \mathbf{r}) = \mathbf{s}$, and update $Table_u[\mathbf{s}]$ based on $Table_a[\mathbf{p}]$ and $Table_b[\mathbf{r}]$. The resulting state for a vertex w_i under $\mathbf{s}$ is computed by addition, using $\oplus$, of subscripts of states under $\mathbf{p}$ and $\mathbf{r}$. Let $size(s)$ denote the subscript of a vertex state s. Moreover, since the neighbors w_i has among $B_b = \{w_1, ..., w_k\}$ are the same in both G_a and G_b we must subtract, using $\ominus$, the shared S-neighbors w_i has in B_b under $\mathbf{p}$ and $\mathbf{r}$. We use the operation $size(p_i) \oplus size(r_i) \ominus c$ defined earlier.

$$combine(\mathbf{p}, \mathbf{r}) := \mathbf{s} \text{ where } \forall i \in \{1, ..., k\} \ \ selected(s_i) = selected(p_i) \text{ and}$$
$$size(s_i) = size(p_i) \oplus size(r_i) \ominus |\{w_q \in B_b : w_i w_q \in E(G) \wedge selected(p_q) = \sigma\}|$$
$$\text{and } s_{k+1} = p_{k+1}$$

We can now give the two-step procedure for the Join operation:

(1) $\forall \mathbf{s} \in I_{k+1} : Table_u[\mathbf{s}] := \bot$;

(2) $\forall (\mathbf{p} \in I_{k+1}, \mathbf{r} \in I_k) :$ if $compatible(\mathbf{p}, \mathbf{r})$ then $\mathbf{s} := combine(\mathbf{p}, \mathbf{r})$ and

$$Table_u[\mathbf{s}] := optimum\{(Table_a[\mathbf{p}] + Table_b[\mathbf{r}]), Table_u[\mathbf{s}]\}$$

where *optimum* is replaced by maximum for a $maxM[L]$-problem and by minimum for a $minM[L]$-problem.

Theorem 6.1 The procedure given for the Join Operation at a node u with children a, b updates $Table_u$ correctly based on correct $Table_a, Table_b$.

Proof. Let u have sources $B_u = \{w_1, ..., w_{k+1}\}$, with notation as before. Consider any $\mathbf{s} = s_1, ..., s_{k+1}$ such that there exists a selected subset of vertices $S \subseteq V(G_u)$ respecting G_u and $\mathbf{s}$, e.g., for $w_i \in B_u$, $Astate_S(w_i) = s_i, 1 \leq i \leq k+1$, with $value = |\{v \in V(G_u) \setminus B_u : Astate_S(v) \in M\}|$. We will show that after executing the Join Table procedure at node u we have $Table_u[\mathbf{s}] \geq value$ for a maximization problem, or $Table_u[\mathbf{s}] \leq value$ for a minimization problem. Let $S \cap V(G_a) = S_A$ and

$S \cap V(G_b) = S_B$. Let $\mathbf{p} = p_1, ..., p_{k+1}$ and $\mathbf{r} = r_1, ..., r_k$ be defined by $p_i = Astate_{S_A}(w_i)$ in G_a and $r_i = Astate_{S_B}(w_i)$ in G_b, respectively. By the assumption that $Table_a$ and $Table_b$ are correct we must have $Table_a[\mathbf{p}] + Table_b[\mathbf{r}] = value$. This since any vertex in $V(G_a) \setminus B_u$ has the exact same state in G_a under S_A as it has in G_u under S, by the fact that there are no adjacencies between a vertex in $V(G_a) \setminus B_u$ and a vertex in $V(G_b) \setminus B_u$. Similarly for G_b. We can check that from the definitions we have $compatible(\mathbf{p}, \mathbf{r}) = 1$ and $combine(\mathbf{p}, \mathbf{r}) = \mathbf{s}$, so indeed $Table_u[\mathbf{s}]$ is updated correctly when the pair $\mathbf{p}, \mathbf{r}$ is considered by the Join procedure.

Now consider an $\mathbf{s}$ such that there does not exist any $S \subseteq V(G_u)$ respecting G_u and $\mathbf{s}$. We will show, by contradiction, that in this case $Table_u[\mathbf{s}]$ is set to $\perp$ initially and then never altered. If $Table_u[\mathbf{s}] \neq \perp$ there must be a compatible pair $\mathbf{p}, \mathbf{r}$ such that $combine(\mathbf{p}, \mathbf{r}) = \mathbf{s}$ and $Table_a[\mathbf{p}] \neq \perp$ and $Table_b[\mathbf{r}] \neq \perp$. Let S_A and S_B be partitions of $V(G_a)$ and $V(G_b)$, respectively, that give these table entries non-$\perp$ values. But then $S = S_A \cup S_B$ would be a subset of $V(G_u)$ respecting G_u such that the resulting state for the separator is $\mathbf{s}$, a contradiction. Again, the reason is that $B_u = \{w_1, ..., w_{k+1}\}$ separates G_u into $G_a \setminus B_u$ and $G_b \setminus B_u$. The above arguments apply to any pair $\mathbf{p}, \mathbf{r}$ considered in the Join operation, and since each such pair updates at most one entry of $Table_u$, we conclude that the Join-Tables operation is correct. □

For each Join node of T the complexity of Join-Tables is $\mathcal{O}(|A|^{2k+1})$ since any pair of entries from tables of children is considered at most once. The procedure for the Join Operation presented in [65] was slightly more complicated in order to avoid consideration of non-compatible pairs. This results in a somewhat better time complexity, but of course, still exponential in k.

Optimize Root Table. Let r be the root of T with $B_r = \{w_1, ..., w_k\}$. We compute $optM[L](G)$ based on correct $Table_r$ as follows

$$optM[L](G) := optimum\{Table_r[\mathbf{s}] + |\{w_i \in B_r : s_i \in M\}|\}$$

where the optimum (min or max) is taken over $\mathbf{s} \in I_k$ such that $s_i \in L, 1 \leq i \leq k$. Correctness of this optimization follows from the definition of table entries and the fact that G_r is the graph G with sources B_r. The complexity of Table-Optimization at the root of T is $\mathcal{O}(|A|^{k+1})$.

6.1.4 Complexity

Correctness of an algorithm based on the given template follows from a simple induction on the parse tree T. As noted in Chapter 5.2, T has $n-k$ Primitive nodes, $n-k$ Reduce nodes and $n-k-1$ Join nodes. Given a tree-decomposition, the algorithm finds the binary parse tree T, executes a single operation at each of its nodes, and performs Table Optimization at the root. The total time complexity becomes $T(n, k, A) = \mathcal{O}(n|A|^{2k+1})$, with Join Tables being the most expensive operation.

Theorem 6.2 Algorithm-optM[L], with A the set of augmented states with respect to L, computes $optM[L](G)$ and has time complexity $T(n, k, A) = \mathcal{O}(n|A|^{2k+1})$.

Corollary 6.1 For any problem $opt[L]$ derived from Table 2.2 ($p \leq 2$) Algorithm-opt[L] has time complexity $T(n, k) = \mathcal{O}(n2^{4k})$.

The corollary follows since any $max[L]$ or $min[L]$ problem over vertex subset properties $[L]$ defined in Table 2.2, with $p \leq 2$, has $|A| \leq 4$. Using the refined procedure for Join-Tables [65] we can get improvements on the overall complexity, the problem Maximum Independent Set achieving complexity $\mathcal{O}(n2^{k+2\log k})$.

6.1.5 Extensions

Our technique applies to a number of more general problems, as follows.

Search Problems. To construct an $[L]$-set of G optimizing the problem parameter we add pointers from each table entry to the table entries of children achieving the optimal value.

Weighted Problems. For weighted versions of the above problems, table entries reflect optimization over the sums of weights of vertices and we need only modify the operations Table Reduce and Table Optimization. The Reduce operation adds the weight of the reduced vertex, when its state is in M, rather than incrementing the optimum sum by one. The Root operation, with the domain of optimization unchanged, becomes

$$optM[L](G) = optimum\{Table_r[\mathbf{s}] + \Sigma weight(w_i) : w_i \in B_r \wedge s_i \in M\}$$

Digraph Problems. For the directed graph versions of these problems we define $IN_G(v) = \{u : \langle u, v\rangle \in$Arcs(G)$\}$ and use $IN_G(v)$, as opposed to $N_G(v)$, in the definition of $Astate_S(v)$, the state of vertex v with respect to a selected set $S \subseteq V(G)$. The only change in the algorithm is for the definition of *combine* in Join-Tables that should use $Arcs(G)$ instead of $E(G)$.

Maximal and Minimal Sets. S is a *maximal* (minimal) $[L]$-set if no vertex can be added to (removed from) S such that the resulting set is still an $[L]$-set. Based on a hand-derived algorithm optimizing over all vertex subsets satisfying some property, Bern *et al* [14] give an automatic procedure constructing an algorithm optimizing over maximal (or minimal) vertex subsets satisfying the same property. This includes an application of Myhill-Nerode finite state automata minimization techniques to minimize the resulting number of separator states. For more on the connection with finite state automata, see also [2]. Unfortunately, when the original algorithm contains

$|A|^k$ separator states, their automatic technique involves simplification of a table with $|A|^k 2^{|A|^k}$ separator states, and quickly becomes infeasible for increasing values of k.

In Chapter 2.3 we give a refined characterization of maximal and minimal vertex subset properties, together with a simple procedure to derive a set of legal vertex states $Lmin$ and $Lmax$ to identify minimal and maximal $[L]$-sets. As an example, S is a minimal dominating set if it is a $[\rho_{\geq 1}, \sigma_0, \sigma_{\geq 1} \cdot \rho_1]$ -set. This extension can be used to design algorithms on partial k-trees for problems which optimize over maximal (or minimal) $[L]$-sets.

We focus on algorithms for minimal vertex subsets, (maximal algorithms defined in an analogous way) and first recall some definitions from Chapter 2.3. With $L = L\rho \cup L\sigma$, let state sets $Amin$, $Bmin$ and $Lmin$ be defined:

$Amin = \{\sigma_i : \sigma_i \in L \wedge \rho_i \notin L\}$ and

$Bmin = \{\rho_i : \rho_i \in L \wedge \rho_{i-1} \notin L\} \cup \{\sigma_i : \sigma_i \in L \wedge \sigma_{i-1} \notin L\}$.

$Lmin = Amin \cup L\rho \cup \{a \cdot b : a \in L\sigma \setminus Amin \wedge b \in Bmin\}$

Theorem 2.1 shows that a vertex subset S is a minimal $[L]$-set in a graph G if and only if it is an $[Lmin]$-set in G (see chapter 2.3 for definitions). In other words, S is a minimal $[L]$-set if and only if S is an $[L]$-set and $\forall v \in S$ either $state_S(v) \in Amin$ or $\exists u \in N_G(v) : state_S(u) \in Bmin$. To account for the latter possibility, we refine the state of vertex v to carry this information about the state of its neighbors. In particular, a vertex with state in $Bmin$ is eligible to become a *mate* of neighboring vertices in S. To design an algorithm solving a problem optimizing over minimal $[L]$-sets we use $Lmin$ as a starting point, find a corresponding set of augmented vertex states and fill in details of Table Initialization, Reduce Table, Join Tables and Root Optimization. For minimal dominating sets, we have $Lmin = \{\rho_{\geq 1}, \sigma_0, \sigma_{\geq 1} \cdot \rho_1\}$ and the augmented states $A = \{\rho_0, \rho_1, \rho_{\geq 2}, \sigma_0, \sigma_{\geq 1}, \sigma_{\geq 0} \cdot \rho_1\}$. We call states using the concatenation operator $\cdot$ a state with a *has*-label, since a vertex with this state is forced to *have* a neighbor of a certain state. A table index containing some state with the *has*-label is initialized to $\perp$. The Table Reduce operation for an index containing states with the *has*-label, is taken as the optimum over table entries of the child whose indices exactly share the *has*-labels, with the added possibility of the reduced vertex having state in the above-defined set $Bmin$ and any neighbor of the reduced vertex having state without the *has*-label. In this latter case, the reduced vertex then becomes the mate of these neighbors. Moreover, the reduced vertex is not allowed to have a σ-state in $L \setminus Amin$ without the *has*-label, as this is not a legal state. The Join Tables operation has the compatibility function altered so that a resulting vertex state with a *has*-label requires the presence of a *has*-label on the corresponding vertex of at least one of the children. The Root Optimization operation considers indices having σ-states in $L \setminus Amin$ without *has*-labels if and only if these are accompanied by a root neighbor with vertex state in $Bmin$.

Irredundant sets. The extensions to our notation outlined above can also be used to design algorithms for many parameters related to *irredundant* sets, see Chapter 2.3.

In this notation, an irredundant set is a $[\rho_{\geq 0}, \sigma_0, \sigma_{\geq 1} \cdot \rho_1]$-set, and the close connection with minimal dominating sets is obvious.

The Irredundance of a graph is the minimum size of a maximal $[\rho_{\geq 0}, \sigma_0, \sigma_{\geq 1} \cdot \rho_1]$-set. Note that a second level of refinement of our characterization is needed to define a set of "doubly" refined vertex states $Imax$ such that the $[Imax]$-sets are exactly the maximal $[\rho_{\geq 0}, \sigma_0, \sigma_{\geq 1} \cdot \rho_1]$-sets. Consequently, the state of a vertex will depend on states of non-neighboring vertices. This has the effect of greatly complicating the design of an algorithm to compute the irredundance number of a partial k-tree, and we do not know any algorithm derived "by hand" for this problem (Bern *et al* [14] solve the problem for trees.)

6.2 Vertex Partitioning Algorithms

In this section we describe algorithms to solve $\exists D_q$-problems, for any degree constraint matrix D_q, see Chapter 2.1. Given an upper bound f for partial k-trees on the parameter in question, a $minD_q$ or $maxD_q$ problem is solved by at most f calls to the $\exists D_q$ algorithm, for different values of q. We call a $maxD_q$ parameter (respectively, a $minD_q$ parameter) *monotone* if existence of a D_q-partition implies the existence of a D_{q-1}-partition (respectively, a D_{q+1}-partition.) For monotone properties we can apply binary search so that $\log f$ calls to the $\exists D_q$ algorithm will suffice.

Algorithm-$\exists D_q$

Input: G, k,tree-decomposition of G of width k

Output: YES if there exists a D_q-partition of $V(G)$, NO o.w.

(1) Find a binary parse tree T of G with Primitive, Reduce and Join nodes.

(2) Initialize Primitive Tables at leaves of T.

(3) Traverse T in the bottom-up manner using Join-Tables and Reduce-Table.

(4) Table-Optimization at root of T gives YES if G has a D_q-partition, NO o.w.

We first discuss the pertinent vertex and separator states and give a description of the tables involved in the algorithm. We then fill in details of table operations, prove their correctness and give their time complexities. Finally, we prove a bound on the Grundy Number of a partial k-tree and adjust the general algorithm template to give us a polynomial-time algorithm for computing that parameter.

6.2.1 Table Description

Our algorithms will follow a binary parse tree of the input graph G. With each node u of T we associate a data structure *table* that stores optimal solutions restricted to G_u, the subgraph of G represented by the subtree of T rooted at u. Each table index at node u represents an equivalence class of solutions to subproblems on G_u, equivalent in terms of being able to form parts of solutions on larger subgraphs. Interaction with larger subgraphs is only through B_u, the sources at u. Each $B_u \subseteq V(G_u)$ is a separator of the given graph G and $|B_u| \in \{k, k+1\}$. For an $\exists D_q$ problem, a solution on G_u is a q-partition of $V(G_u)$. The equivalence relation on solutions is defined as follows:

> **Definition 6.8** Two partitions P_1 and P_2 of $V(G_u)$ are equivalent if augmenting G_u to G, with the new vertices classified as belonging to some of the partition classes has the effect that either both or none of the thus augmented P_1' and P_2' are D_q-partitions of the new graph.

Based on the above definition, we can define equivalence classes of solutions by identifying a *separator state* with each equivalence class. States for a separator B_u are in turn defined as $|B_u|$-vectors of *vertex states*. The state of vertex v under partition $V_1, ..., V_q$ encodes the effect the partition has on v. To capture the equivalence relation in Definition 6.8, our algorithms will need an *augmented* degree matrix A_q, from which we derive the augmented vertex states A. Comparing with vertex subset problems, the degree constraint matrix D_q replaces D_ρ, D_σ from Definition 6.1 and A_q replaces $\alpha(M_\rho, D_\rho), \alpha(M_\sigma, D_\sigma)$ from Definition 6.3. We first define a function $\beta : 2^{\mathbb{N}} \rightarrow R$ (recall Definition 6.2) such that $A_q[i,j] = \beta(D_q[i,j])$.

> **Definition 6.9** $A_q[i,j] = \beta(D_q[i,j])$ where

$$\beta(D_q[i,j]) = \begin{cases} Y_t & \text{if } \exists t \in D_q[i,j] \text{ s.t. } t = \max\{D_q[i,j]\} \\ W_t & \text{if } \exists t \in D_q[i,j] \text{ with } t \text{ minimum s.t. } \{t, t+1, ...\} \subseteq D_q[i,j] \\ \mathbb{N} & \text{otherwise} \end{cases}$$

The augmented vertex states A are defined by focusing on rows of A_q. An augmented vertex state will consist of a pair $(i)(M)$ where $1 \leq i \leq q$ indexes a row of A_q and M is an element of the Cartesian product $A_q[i,1] \times A_q[i,2] \times ... \times A_q[i,q]$

> **Definition 6.10** For an $\exists D_q$ problem we define the augmented vertex state set:
>
> $A = \{(i)(M_{i1}M_{i2}...M_{iq}) \ : i \in \{1,...,q\} \wedge \forall j(j \in \{1,...,q\} \Rightarrow (M_{ij} \in \beta(D_q[i,j])))\}$

Note that our algorithmic template will not work if $A_q[i,j] = \mathbb{N}$ for any entry of A_q, as this would imply an infinite vertex state set. We consider an example. Figure 6.1 shows the matrix D_3 such that the $\exists D_3$ problem decides whether vertices of a graph

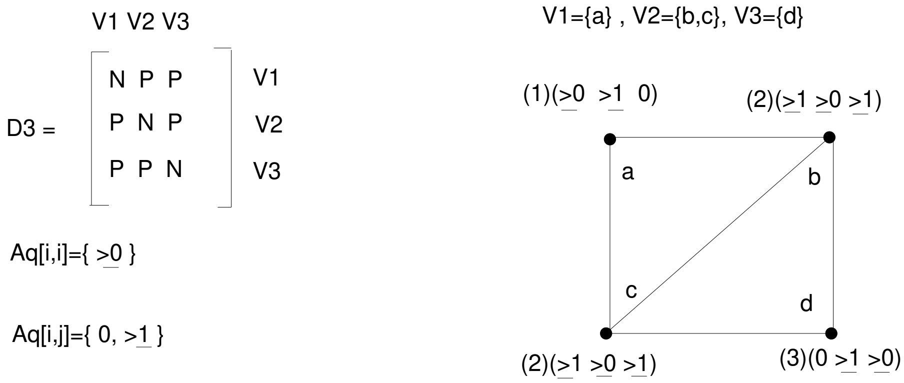


Figure 6.1: The matrix D_3 for deciding whether there exists a partition into 3 dominating sets (N=$\{0, 1, ...\}$ and P=$\{1, 2, ...\}$). Also, resulting states for a given partition on a graph. Note that vertices b and c satisfy the constraints given by D_3, as can be seen from comparing their states with row 2 of D_3. Vertices a and d need additional neighbors if this partition is to be augmented to a D_3-partition of some supergraph.

can be partitioned into 3 dominating sets. Diagonal entries of D_3 are $\mathbb{N}$ and off-diagonal entries are $\mathbb{P}$. Applying Definition 6.9 above we get $\beta(D_q[i,i]) = W_0 = \{\geq 0\}$ and $\beta(D_q[i,j]) = W_1 = \{0, \geq 1\}$ for $i \neq j$. For this problem we then get the 12 vertex states:

$$\begin{array}{llll}
\{(1)(\geq 0\ \ 0\ \ 0), & (1)(\geq 0\ \ 0\ \ \geq 1), & (1)(\geq 0\ \ \geq 1\ \ 0), & (1)(\geq 0\ \ \geq 1\ \ \geq 1),\\
\ (2)(0\ \ \geq 0\ \ 0), & (2)(0\ \ \geq 0\ \ \geq 1), & (2)(\geq 1\ \ \geq 0\ \ 0), & (2)(\geq 1\ \ \geq 0\ \ \geq 1),\\
\ (3)(0\ \ 0\ \ \geq 0), & (3)(0\ \ \geq 1\ \ \geq 0), & (3)(\geq 1\ \ 0\ \ \geq 0), & (3)(\geq 1\ \ \geq 1\ \ \geq 0)\}
\end{array}$$

The three states at the rightmost column above are the legal states, corresponding to the three rows of the degree constraint matrix D_3. For a partition $V_1, V_2, ..., V_q$ of $V(G)$ and a vertex $v \in V(G)$ we use the natural definition of $A_q state(v)$ arising from the augmented degree constraint matrix A_q :

$$A_q state_{V_1,...,V_q}(v) = \begin{cases} (1)(\geq 0\quad 0\quad 0) & \text{if } v \in V_1 \text{ and } |N_G(v) \cap V_1| \geq 0 \wedge \\ & \quad \wedge |N_G(v) \cap V_2| = 0 \wedge |N_G(v) \cap V_3| = 0 \\ ... & \\ (3)(\geq 1\ \ \geq 1\ \ \geq 0) & \text{if } v \in V_3 \text{ and } |N_G(v) \cap V_1| \geq 1 \wedge \\ & \quad \wedge |N_G(v) \cap V_2| \geq 1 \wedge |N_G(v) \cap V_3| \geq 0 \end{cases}$$

This extends to 12^k separator states, of a separator of size k, in the partial k-tree algorithm deciding whether there exists a partition into 3 dominating sets.

We return to discussing the general $\exists D_q$-algorithm and examine the size of the augmented vertex state set A and the index set of the table I_k at a node u of the parse tree with k sources. Assume for simplicity that the matrix D_q has diagonal entries $L_\sigma \subseteq \mathbb{N}$ and off-diagonal entries $L_\rho \subseteq \mathbb{N}$. Let $A_\sigma = A_q[i,i] = \beta(D_q[i,i])$ and

$A_\rho = A_q[i,j] = \beta(D_q[i,j])$ for $i \neq j$. Note that $A_q state(v) = (i)(M_{i1}M_{i2}...M_{iq})$ with $i \in \{1,2,...,q\}$, $M_{ii} \in A_\sigma$ and $M_{ij} \in A_\rho$ for $i \neq j$. With A the set of augmented vertex states for the $\exists D_q$-problem, we thus have $|A| = q|A_\sigma||A_\rho|^{q-1}$. Consider a node u of the parse tree. Let $B_u = \{w_1, w_2, ..., w_k\}$ with $Table_u$ having index set $I_k = \{\mathbf{s} = s_1, ..., s_k\}$ for any $s_i \in A$. Thus the size of the table is $|I_k| = |A|^k$.

We now turn to the values of table entries. Define Ψ with respect to G_u and $\mathbf{s} = s_1, ..., s_k\}$, $s_i \in A$, to be the family of partitions $V_1, V_2, ..., V_q$ of $V(G_u)$, such that in G_u, for $w_i \in B_u$, $A_q state_{V_1,V_2,...,V_q}(w_i) = s_i$ and for $v \in V(G_u) \setminus B_u$ if $v \in V_i$ then $|N_G(v) \cap V_j| \in D_q[i,j], j = 1, ..., q$.

Definition 6.11 For problem $\exists D_q$ with augmented vertex states A, graph G_u with sources $B_u = \{w_1, w_2, ..., w_k\}$ and k-vector $\mathbf{s} = s_1, ...s_k : s_i \in A$ we define
$\Psi \overset{df}{=} \{V_1, ..., V_q$ a q-partition of $V(G_u)$ $: \forall v \in V(G_u) \setminus B_u \ \forall w_i \in B_u$
$A_q state_{V_1,...,V_q}(v) \in D_q[i,j]$ and $A_q state_{V_1,...,V_q}(w_i) = s_i\}$

Ψ forms an equivalence class of solutions to the subproblem on G_u, and its elements are called Ψ-partitions respecting G_u and $\mathbf{s}$. Note that states of sources are allowed to be any augmented vertex state, whereas an "eliminated" vertex must have state as constrained by D_q. The binary contents of $Table_u[s]$ records whether any solution respecting G_u and $\mathbf{s}$ exists:

Definition 6.12

$$Table_u[\mathbf{s}] = \begin{cases} 1 & \text{if } \Psi \neq \emptyset \\ 0 & \text{if } \Psi = \emptyset \end{cases}$$

6.2.2 Table Operations

We now elaborate on the operations of Table-Initialization, Table-Reduce, Join-Tables and Table-Root-Optimization. Each of the following subsections defines the appropriate procedure, gives the proof of its correctness and analyzes its complexity.

<u>Table-Initialization</u>

A leaf u of T is a Primitive node and G_u is the graph $G[B_u]$, where $B_u = \{w_1, ..., w_{k+1}\}$. Let $Partition(B_u)$ be all partitions of B_u into distinguished partition classes $V_1, ..., V_q$. Following the given definition of tables we initialize $Table_u$ in two steps

(1) $\forall \mathbf{s} \in I_{k+1} : Table_u[\mathbf{s}] := 0$
(2) $\forall V_1, V_2, ..., V_q \in Partition(B_u)$: if $V_1, ..., V_q$ is a D_q-partition of $G[B_u]$ with $A_q state_{V_1,...,V_q}(w_i) = s_i, i = 1, ..., k+1$, then for $\mathbf{s} = s_1, ..., s_{k+1}$ $Table_u[\mathbf{s}] := 1$

The complexity of this initialization for each leaf of T is $\mathcal{O}(|I_{k+1}| + kq^{k+1})$.

Reduce Table

A Reduce node u of T has a single child a such that $B_u = \{w_1, ..., w_k\}$ and $B_a = \{w_1, ..., w_{k+1}\}$. We compute $Table_u$ based on correct $Table_a$ as follows

$$\forall \mathbf{s} \in I_k : Table_u[\mathbf{s}] := OR \quad \{Table_a[\mathbf{p}]\}$$

where the OR is over all $\mathbf{p} \in I_{k+1}$ with $\forall l : 1 \leq l \leq k, p_l = s_l$ and $p_{k+1} = (i)(M_{i1}, ..., M_{iq})$ such that for $j = 1, ..., q$ we have $M_{ij} \in D_q[i, j]$ or $M_{ij} = D_q[i, j]$. Correctness of the operation follows by noting that G_a and G_u designate the same subgraph of G, and differ only by w_{k+1} not being a source in G_u. By definition, $Table_u[\mathbf{s}]$ should store a 1 iff there is some Ψ-set respecting G_a and $\mathbf{s}$ where the state of non-sources, *e.g.* w_{k+1}, is constrained by D_q. The complexity of this operation for each Reduce node of T is $\mathcal{O}(|I_{k+1}|)$.

Join Tables

A Join node u of T has children a and b such that $B_u = B_a = \{w_1, ..., w_{k+1}\}$ and $B_b = \{w_1, ..., w_k\}$ is a k-subset of B_a. Moreover, G_a and G_b share exactly the subgraph induced by B_b, $G[B_b]$. We compute $Table_u$ by considering all pairs of table entries of the form $Table_a[\mathbf{p}], Table_b[\mathbf{r}]$. Recall that the separator state $\mathbf{p}$ consists of $k+1$ vertex states $p_1, p_2, ..., p_{k+1}$ where the state p_i is associated with vertex w_i. A vertex state consists of the partition class index $class(p_i)$ of w_i, *i.e.* $w_i \in V_{class(p_i)}$, and a q-vector denoting the cardinality, expressed as an element of the Cartesian product $A_q[i, 1] \times A_q[i, 2] \times ... \times A_q[i, q]$, of w_i's neighborhood in each partition class $V_1, ..., V_q$. We use the notation $size(p_i, j)$ to denote the jth component of this q-vector, *i.e.* the size of w_i's neighborhood in V_j, as specified by p_i. In the procedure for the Join operation, we first check that $\mathbf{p}, \mathbf{r}$ is a *compatible* separator state pair, meaning the partition class assigned to vertex $w_i, i \in \{1, ..., k\}$ is identical in both $\mathbf{p}$ and $\mathbf{r}$.

$$compatible(\mathbf{p}, \mathbf{r}) := \begin{cases} 1 & \text{if } class(p_i) = class(r_i)\ \forall i \in \{1, ..., k\} \\ 0 & \text{otherwise} \end{cases}$$

We then combine, for each $w_i, i \in \{1, ..., k+1\}$ the contributions from $\mathbf{p}$ and $\mathbf{r}$ to give the resulting separator state $combine(\mathbf{p}, \mathbf{r}) = \mathbf{s}$, and update $Table_u[\mathbf{s}]$ based on $Table_a[\mathbf{p}]$ and $Table_b[\mathbf{r}]$. The resulting q-vector of neighborhood sizes for a vertex w_i under $\mathbf{s}$ is computed by (componentwise) addition of its q-vectors under $\mathbf{p}$ and $\mathbf{r}$. This addition at the jth component is performed using $\oplus$ from Definition 6.5. Moreover, since the neighbors w_i has in $B_b = \{w_1, ..., w_k\}$ are the same in both G_a and G_b we must subtract the shared V_j neighbors w_i has in B_b under $\mathbf{p}$ and $\mathbf{r}$. We thus use

$$combine(\mathbf{p}, \mathbf{r}) := \mathbf{s} \text{ where } \forall i \in \{1, ..., k\}\ \ \forall j \in \{1, ..., q\}$$

$$class(s_i) = class(r_i) = class(p_i) \text{ and}$$

$$size(s_i, j) = size(p_i, j) \oplus size(r_i, j) \ominus |\{w_l \in B_b : w_i w_l \in E(G) \wedge class(p_l) = j\}|$$

$$\text{and } s_{k+1} = p_{k+1}$$

We can now state the two step procedure for the Join operation:

(1) $\forall \mathbf{s} \in I_{k+1} : Table_u[\mathbf{s}] := 0;$

(2) $\forall(\mathbf{p} \in I_{k+1}, \mathbf{r} \in I_k)$: if $compatible(\mathbf{p}, \mathbf{r})$ and $Table_a[\mathbf{p}] = Table_b[\mathbf{r}] = 1$

then $Table_u[combine(\mathbf{p}, \mathbf{r})] := 1$

Theorem 6.3 The procedure given for the Join Operation at a node u with children a, b updates $Table_u$ correctly based on correct $Table_a, Table_b$.

Proof. We argue the correctness of the Join operation at a node u with sources $B_u = \{w_1, ..., w_{k+1}\}$, based on correct table entries at its children a and b, with notation as before. Consider any $\mathbf{s} = s_1, ..., s_{k+1}$ such that there exists a partition $V_1, ..., V_q$ of $V(G_u)$ respecting D_q with $A_q state_{V_1,...,V_q}(w_i) = s_i$ for $i = 1$ to $k+1$ in the graph G_u. We will show that then $Table_u[\mathbf{s}]$ is correctly set to the value 1. Let $A_1, ..., A_q$ and $B_1, ..., B_q$ be the induced partitions on $V(G_a)$ and $V(G_b)$, respectively, *i.e.*, $V_i \cap V(G_a) = A_i$ and $V_i \cap V(G_b) = B_i$. Let $\mathbf{p} = p_1, ..., p_{k+1}$ and $\mathbf{r} = r_1, ..., r_k$ be defined by $p_i = A_q state_{A_1,...,A_q}(w_i)$ in G_a and $r_i = A_q state_{B_1,...,B_q}(w_i)$ in G_b, respectively. By the assumption that $Table_a$ and $Table_b$ are correct we must have $Table_a[\mathbf{p}] = Table_b[\mathbf{r}] = 1$. This since any vertex in $V(G_a) \setminus B_u$ has the exact same state in G_a under $A_1, ..., A_q$ as it has in G_u under $V_1, ..., V_q$, by the fact that there are no adjacencies between a vertex in $V(G_a) \setminus B_u$ and a vertex in $V(G_b) \setminus B_u$. Similarly for G_b. We can check that from the definitions we have $compatible(\mathbf{p}, \mathbf{r}) = 1$ and $combine(\mathbf{p}, \mathbf{r}) = \mathbf{s}$, so indeed $Table_u[\mathbf{s}]$ is set to 1 when the pair $\mathbf{p}, \mathbf{r}$ is considered by the algorithm.

Now consider an $\mathbf{s}$ such that there does not exist any q-partition of $V(G_u)$ respecting D_q such that the resulting state for the separator is $\mathbf{s}$. We will show, by contradiction, that in this case $Table_u[\mathbf{s}]$ is set to 0 initially and then never altered. If $Table_u[\mathbf{s}] = 1$ there must be a compatible pair $\mathbf{p}, \mathbf{r}$ such that $combine(\mathbf{p}, \mathbf{r}) = \mathbf{s}$ and $Table_a[\mathbf{p}] = Table_b[\mathbf{r}] = 1$. Let $A_1, ..., A_q$ and $B_1, ..., B_q$ be partitions of $V(G_a)$ and $V(G_b)$, respectively, that set these table entries to 1. Then $V_1, ..., V_q$ defined by $V_i = A_i \cup B_i$ is a q-partition of $V(G_u)$ respecting D_q such that the resulting state for the separator is $\mathbf{s}$, because $B_u = \{w_1, ..., w_{k+1}\}$ separates G_u into $G_a \setminus B_u$ and $G_b \setminus B_u$. This contradicts our assumption that such a q-partition does not exist. We conclude that the Join-Tables operation is correct. □

For each Join node of T the complexity of Join-Tables is $\mathcal{O}(|I_k||I_{k+1}|)$ since any pair of entries from tables of children is considered at most once.

Optimize Root Table

Let r be the root of T with $B_r = \{w_1, ..., w_k\}$. We decide whether G has a D_q-partition based on correct $Table_r$ as follows

YES if $\exists \mathbf{s} = s_1, ..., s_k \in I_k$ such that $Table_r[\mathbf{s}] = 1$ and for $1 \leq i \leq k, 1 \leq j \leq q$ we have $s_i = (x)(M_{x1}, ..., M_{xq})$ with $M_{xj} \in D_q[x, j]$

NO otherwise

Correctness of this optimization follows from the definition of table entries and the fact that G_r is the graph G with sources B_r. The complexity of Table optimization at the root of T is $\mathcal{O}(|I_{k+1}|)$.

6.2.3 Overall Correctness and Complexity

Correctness of an algorithm based on this algoritmic template follows by induction on the binary parse tree T. As noted in Chapter 5.2, T has $n-k$ Primitive nodes, $n-k$ Reduce nodes and $n-k-1$ Join nodes. The algorithm finds the binary parse tree T, executes a single respective operation at each of its nodes, and performs Table Optimization at the root.

> **Theorem 6.4** The time complexity for solving an $\exists D_q$ problem with augmented vertex state set A, table index set I_k for k sources, on a partial k-tree with n vertices, given a tree-decomposition of width k is
>
> - $T(n,k) = \mathcal{O}(n|I_k||I_{k+1}|)$
> - $T(n,k,q) = \mathcal{O}(n|A|^{2k+1})$.
> - $T(n,k,q,A_\sigma,A_\rho) = \mathcal{O}(nq^{2k+1}|A_\sigma|^{2k+1}|A_\rho|^{(2k+1)(q-1)})$

Proof. The first bound follows since Join Tables is the most expensive operation. The next two bounds come from $|I_k| = |A|^k$ and $|A| = q|A_\sigma||A_\rho|^{q-1}$. □

Note that the last bound is stated for problems where D_q has all diagonal entries equal to A_σ and all off-diagonal entries equal to A_ρ. For more general vertex partitioning problems, we simply take $|A_\sigma| = \max_i\{\beta(D_q[i,i])\}$ and $|A_\rho| = \max_{i\neq j}\{\beta(D_q[i,j])\}$ to be the largest augmentations resulting from D_q anywhere on and off the diagonal, respectively. For $maxD_q$ and $minD_q$ problems we get algorithms linear in n if the optimized parameter is bounded from above on partial k-trees by a function of k only. This is the case for both chromatic number and domatic number which have the bound $k+1$ on partial k-trees. Since these properties are also monotonic, as discussed earlier, we can do a binary search for the correct value with $\log k + 1$ calls to an $\exists D_q$ problem. Resulting time bounds for specific problems are shown in Table 6.1, as discussed also in the following sections.

6.2.4 Grundy Number Algorithm

Computing the Grundy number of an undirected graph is NP-complete even for bipartite graphs and for chordal graphs [55]. A binomial tree on 2^{q-1} vertices has Grundy number q [41] and in general the non-existence of an $f(k)$ upper bound on the Grundy number of a partial k-tree explains the lack of a description of this problem in EMSOL [50]. For trees there exists a linear time algorithm [41] but until now it was an open question whether polynomial time algorithms existed even for 2-trees [38]. Recall from Chapter 2.1 that the definition of Grundy number as a vertex

Problem	q	$\lvert A_\sigma\rvert$	$\lvert A_\rho\rvert$	Time Complexity
CHROMATIC NUMBER	$1 \leq q \leq k+1$	1	1	$\mathcal{O}(nk^{2(k+1)})$
q-COLORING	q	1	1	$\mathcal{O}(nq^{2(k+1)})$
H-COVER	$q = \lvert V(H)\rvert$	1	2	$\mathcal{O}(n2^{3k\lvert V(H)\rvert})$
H-COLOR	$q = \lvert V(H)\rvert$	1	1	$\mathcal{O}(n\lvert V(H)\rvert^{2(k+1)})$
DOMATIC NUMBER	$1 \leq q \leq k+1$	1	2	$\mathcal{O}(n2^{3k^2})$
DISTANCE $\leq q$ DOM.	$q+1$	1	2	$\mathcal{O}(n2^{3k(q+1)})$
GRUNDY NUMBER	$1 \leq q \leq 1 + k\log n$	1	2	$\mathcal{O}(n^{3k^2})$
UPPER DOM. REMOVAL	$1 \leq q \leq 1 + k\log n$	1	2	$\mathcal{O}(n^{3k^2})$

Table 6.1: Time complexity for specific problems on partial k-trees of n vertices

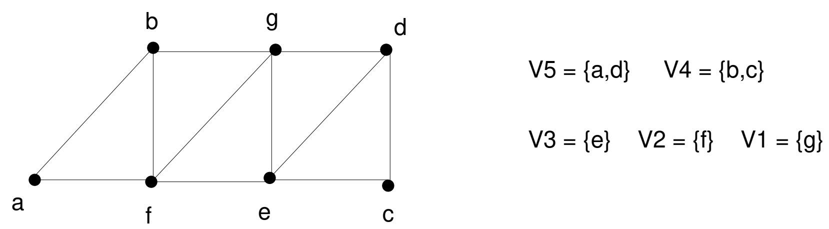


Figure 6.2: A 2-tree on 7 vertices with Grundy number 5 and an appropriate partition $V1, V2, ..., V5$

partitioning problem required all partition classes to be non-empty. In this section we first show how the algorithm template of section 6.2.2 can be easily adjusted to enforce this requirement. We also prove a logarithmic, in $|V(G)|$, upper bound on the Grundy Number, $GN(G)$, of a partial k-tree G. These results suffice to show the polynomial time complexity of computing the Grundy number of any partial k-tree, for fixed k.

To facilitate the presentation of these results, we reverse the ordering of the partition classes in the definition of GN from Chapter 2.1; this is expressed by the degree constraint matrix D_q with diagonal entries $\{0\}$, above-diagonal entries $\mathbb{P}$, and below-diagonal entries $\mathbb{N}$. Thus, for a graph G, $GN(G)$ is the largest value of q such that its vertices $V(G)$ can be partitioned into non-empty classes $V_1, V_2, ..., V_q$ with the constraint that for $i = 1, ..., q$, V_i is an independent set and every vertex in V_i has at least one neighbor in each of the sets $V_{i+1}, V_{i+2}, ..., V_q$ (see Figure 6.2.) Note that if we have at least one vertex $v \in V_1$ then this guarantees that every partition class is non-empty, since D_q requires v to have at least one neighbor in each of $V_2, V_3, ..., V_q$. In the algorithm for deciding whether a partial k-tree has a D_q-partition with non-empty classes, with D_q as described above, we augment the value of a table entry $Table_u[\mathbf{s}]$ by a single extra bit called *nonempty*. This bit will record whether there exists any partition $V_1, ..., V_q$ respecting G_u and the separator state $\mathbf{s}$ such that $V_1 \neq \emptyset$. In the following, we use notation as given in section 6.2.2, with the definition of table entries:

$$Table_u[\mathbf{s}] = \begin{cases} \langle 0,0\rangle & \text{if } \Psi = \emptyset \\ \langle 1,0\rangle & \text{if } \Psi \neq \emptyset \text{ but } \not\exists V_1, V_2, ..., V_q \in \Psi \text{ with } V_1 \neq \emptyset \\ \langle 1,1\rangle & \text{if } \Psi \neq \emptyset \text{ and } \exists V_1, V_2, ..., V_q \in \Psi \text{ with } V_1 \neq \emptyset \end{cases}$$

The two-step Table-Initialization procedure becomes:

(1) $\forall \mathbf{s} \in I_{k+1} : Table_u[\mathbf{s}] := \langle 0,0\rangle$

(2) $\forall V_1, V_2, ..., V_q \in Partition(B_u)$: if $V_1, ..., V_q$ is a D_q-partition of $G[B_u]$ with

$\mathbf{s} = s_1, ..., s_{k+1}$ such that $A_q state_{V_1,...,V_q}(w_i) = s_i, i = 1, ..., k+1$ then

if $V_1 = \emptyset$ set $Table_u[\mathbf{s}] := \langle 1,0\rangle$

else if $V_1 \neq \emptyset$ set $Table_u[\mathbf{s}] := \langle 1,1\rangle$

Note that for a leaf u of the binary parse tree of G, all vertices of G_u are sources so the separator state $\mathbf{s}$, in step (2) above, contains the information determining if V_1 is empty. The Reduce-Table procedure remains as given in section 6.2.2 except that the OR is taken over both bits in the values of table entries, *i.e.*, $\langle a,b\rangle \vee \langle c,d\rangle = \langle a \vee c\rangle, \langle b \vee d\rangle$. For the Join-Table procedure, the concepts of compatibility and combining of pairs are unchanged, whereas the two-step update procedure becomes:

(1) $\forall \mathbf{s} \in I_{k+1} : Table_u[\mathbf{s}] := \langle 0,0\rangle$;

(2) $\forall (\mathbf{p} \in I_{k+1}, \mathbf{r} \in I_k)$: if *compatible*$(\mathbf{p},\mathbf{r})$ and

$Table_a[\mathbf{p}] = \langle 1,x\rangle$ and $Table_b[\mathbf{r}] = \langle 1,y\rangle$ and $Table_u[combine(\mathbf{p},\mathbf{r})] = \langle z,w\rangle$

then $Table_u[combine(\mathbf{p},\mathbf{r})] := \langle 1, x \vee y \vee w\rangle$.

Root optimization becomes:

YES if $\exists \mathbf{s} = s_1, ..., s_k \in I_k$ such that $Table_r[\mathbf{s}] = \langle 1,1\rangle$ and for $1 \leq i \leq k, 1 \leq j \leq q$

we have $s_i = (x)(M_{x1}, ..., M_{xq})$ with $M_{xj} \in D_q[x,j]$

NO otherwise

It is easy to see that the time complexity of the resulting algorithm remains as described by Theorem 6.4.

We now turn to the bound on the Grundy number $GN(G)$ of a partial k-tree G. Since the Grundy number of a graph may increase by removing edges we cannot restrict attention to k-trees, but must consider partial k-trees. A tree (*i.e.* a 1-tree) with Grundy number q, witnessed by a (Grundy) partition $V_1, ..., V_q$, must have at least 2^{q-1} vertices since a vertex $v_1 \in V_1$ must have a neighbor $v_2 \in V_2$, both v_1, v_2 must have (distinct) neighbors $v_3, v_4 \in V_3$, vertices $v_1, ..., v_4$ must have neighbors

$v_5, ..., v_8 \in V_4$, etc. (more precisely, each vertex of the set $\bigcup_{1 \leq i < j} V_i$ has a unique neighbor in V_j thus doubling the size of $\bigcup_{1 \leq i \leq j} V_i$ for each consecutive $1 < j \leq q$.) This argument relies on the fact that 1-trees do not have cycles. For a partial k-tree G with $k \geq 2$ and Grundy number q we cannot guarantee the existence of a perfect elimination ordering ($peo = v_n, v_{n-1}, ..., v_1$) of vertices which respects a $V_q, ..., V_1$ Grundy partition of $V(G)$, as in the 1-tree example above. See Figure 6.2 for an example of a 2-tree on 7 vertices which does not have a perfect elimination ordering respecting the partial order given by any Grundy partition $V_5, V_4, ..., V_1$. Hence, the general bound given below has a somewhat less trivial proof than the 1-tree case.

Theorem 6.5

For G a partial k-tree on $n \geq k \geq 1$ vertices, we have

$$GN(G) \leq 1 + \lfloor \log_{(k+1)/k} n \rfloor$$

Proof. Let $GN(G) = q$ with $V_1, V_2, ..., V_q$ an appropriate partition of $V(G)$ as described above. For $1 \leq i \leq q$, define G_i to be the graph $G \setminus (\cup V_j, j > i)$. Thus $G_q = G$ and in general G_i is the graph induced by vertices $V_1 \cup V_2 ... \cup V_i$ with V_i a dominating set of G_i. Let $n_i = |V(G_i)|$ and $m_i = |EG_i)|$. By induction on i from k to q we show that in this range

$$n_i \geq (\frac{k+1}{k})^{i-1}$$

For the base case $i = k$ we have $(2/1)^0 \leq 1 \leq n_1$ and $(3/2)^1 < 2 \leq n_2$ and for $k \geq 3$ $(1 + 1/k)^{k-1} \leq (1 + 1/k)^k \leq e < 3 \leq n_k$. Note that the inequality is strict for $k \geq 2$. We continue with the inductive step of the proof, with the inductive assumption that the inequality holds for j in the range k to $i-1$ and establish the inequality for $j = i$. Note that $m_i - m_{i-1}$ counts the number of edges in G_i with at least one endpoint in V_i. Since every vertex in $V(G_{i-1}) = V_1 \cup V_2 \cup ... \cup V_{i-1}$ has at least one G_i-neighbor in V_i we get a lower bound on m_i

$$m_i \geq m_{i-1} + n_{i-1}$$

G_i is a subgraph of a k-tree, and if $i \geq k$ then it is a partial k-tree on $n_i \geq k$ vertices. It is well-known that G_i is then a subgraph of a k-tree on n_i vertices [6], and from the iterative construction of k-trees it is easy to show that we have

$$m_i \leq \frac{k(k-1)}{2} + (n_i - k)k$$

Rearranging terms we get the following bound on n_i for $k \leq i \leq q$

$$n_i \geq \frac{m_i}{k} + \frac{k+1}{2}$$

Repeatedly substituting the m_i bound in the above, we get

$$n_i \geq \frac{m_{i-1} + n_{i-1}}{k} + \frac{k+1}{2} \geq ... \geq \frac{n_k + n_{k+1} + ... + n_{i-2} + n_{i-1}}{k} + \frac{m_k}{k} + \frac{k+1}{2}$$

In the right-hand side we substitute for all n_j the inductive bound $n_j \geq (\frac{k+1}{k})^{j-1}$ to get

$$n_i \geq \frac{1}{k} \sum_{j=k-1}^{i-2} (\frac{k+1}{k})^j + \frac{m_k}{k} + \frac{k+1}{2} = (\frac{k+1}{k})^{i-1} - (\frac{k+1}{k})^{k-1} + \frac{m_k}{k} + \frac{k+1}{2}$$

Since V_j is a dominating set in G_j for $1 \leq j \leq k$ we must have $m_k \geq (k-1)k/2$ which we substitute in the above to get the desired bound

$$n_i \geq (\frac{k+1}{k})^{i-1} - (\frac{k+1}{k})^{k-1} + k \geq (\frac{k+1}{k})^{i-1}$$

Note that the last bound is strict for $k \geq 2$(*). For $i = q$ we thus get $q \leq 1 + \log_{(k+1)/k} n_q$ (note that $q = GN(G)$ and $n_q = n$) which is a tight bound for $k = 1$. For $k \geq 2$, the base is not an integer and, because of the strict inequality (*) we can apply the floor function to the log. $\square$

For fixed k we thus have a logarithmic bound on GN for partial k-trees. Since we want to express time complexity as a function of k, we convert bases of the logarithm to get $GN(G) \leq 1 + (\log_2 \frac{k+1}{k})^{-1} \log_2 n \leq 1 + k \log_2 n$.

> **Theorem 6.6** Given a partial k-tree G on n vertices its Grundy number can be found in $\mathcal{O}(n^{3k^2})$ time.

Proof. First note that a tree-decomposition can be found in time linear in n [17]. Define the Grundy number problem using the degree constraint matrix D_q with diagonal entries $\{0\}$, above-diagonal entries $\mathbb{P}$, and below-diagonal entries $\mathbb{N}$. We then use the algorithm from section 6.2.2 augmented with the *nonempty* information as described above. The correctness of each table operation procedure is easily established, so that by induction over the parse tree we can conclude that the root-optimization procedure will correctly give the answer YES if and only if the input graph has an appropriate partition $V_1, ..., V_q$ with non-empty classes. An affirmative answer implies that $GN(G) \geq q$. Using the bound $GN(G) \leq 1 + k \log_2 n$ we run the $\exists D_q$ algorithm for descending values of q starting with $q = 1 + k \log_2 n$ and halting as soon as an affirmative answer is given. The complexity of this algorithm is then given by appropriately applying Theorem 6.4, with $|A_\sigma| = 1$ and $|A_\rho| = 2$. $\square$

6.2.5 Extensions

Algorithms for search versions, weighted versions and directed graph versions of vertex partitioning problems are constructed by the same general method as the extensions given for vertex subset optimization problems in section 6.1.5.

In chapter 2.1 we discussed several problems which could be defined by allowing optimization over the cardinality of certain partition classes. For example, the DISTANCE $\leq q$ DOMINATION problem optimizes $|V_1|$ over all D_{q+1}-partitions $V_1, V_2, ..., V_{q+1}$ for the appropriate degree constraint matrix D_{q+1}. To compute this parameter, the values of table entries are defined to be

$$Table_u[\mathbf{s}] \stackrel{df}{=} \begin{cases} \perp & \text{if } \Psi = \emptyset \\ optimum_{V_1,...V_{q+1} \in \Psi}\{|V_1|\} & \text{otherwise} \end{cases}$$

and the table operations are altered similarly, in the style of table operations for vertex subset optimization problems. Any vertex partitioning problem optimizing over the cardinality of a partition class can be solved in a similar manner. The time complexity of the resulting algorithms for a problem given by the degree constraint matrix D_q remains as given in Theorem 6.4. For example, the DISTANCE $\leq q$ DOMINATION problem is solved on partial k-trees, when given a tree-decomposition, in time $\mathcal{O}(n2^{3k(q+1)})$.

In Chapter 2.4 we discussed several new problems, including the general classes of $[\rho, \sigma]$-PARTITION problems, NON-UNIFORM PARTITION problems and $[\rho, \sigma]$-REMOVAL problems. Any of these problems are encompassed by Theorem 6.4. Polynomial-time algorithms for $maxD_q$ and $minD_q$ problems will of course only follow if an appropriate bound holds on the parameter in question. In particular, note that the proof of the logarithmic bound on the Grundy Number in Theorem 6.5 does not utilize the fact that the V_i are independent sets, only the fact that they are dominating sets in the remaining graph. This means we get a logarithmic bound also on the UPPER-DOMINATING-REMOVAL parameter on partial k-trees (see Chapter 2.4 for a definition) and a polynomial time algorithm for computing this parameter, for fixed k.

Chapter 7

Conclusions

Several parts of this thesis are based on previously published work:

- Sections 2.2, 2.3 and 2.5 are based on [63]
- Chapter 3 is based on [64]
- Chapter 4 is based on [48]
- Sections 5.1, 5.2, 5.3 and 6.1 are based on [65]

We discuss some natural continuations of the work presented.

We hope that the vertex partitioning characterization given in chapter 2 will open several avenues of future work.

As an example we mention the search for general results on tractability/intractability cutoff points for the various vertex partitioning problems, where for example 2-coloring is easy while 3-coloring is $\mathcal{NP}$-complete. For many vertex partitioning problems this question has not yet been explored.

We also plan to investigate the fixed parameter tractability of vertex partitioning problems, giving membership and hardness results for the W hierarchy [18].

An obvious continuation of the work in Chapter III is to complete the classification of the complexity of vertex subset problems for general graphs. We have several conjectures in this direction awaiting further investigation.

In a related effort, Kratochvíl *et al.* [47] employ our characterization of vertex subset problems from Chapter II to study their complexity on both interval graphs and chordal graphs.

The current joint work on graph covering problems in Chapter IV has already proven to be a fertile field. The ultimate goal is to completely classify the computational complexity of H-covering problems for any graph H, and again we have several conjectures to work on.

Regarding partial k-tree algorithms, there are several possibilities of generalizing our results. A possibly easy extension would be to encompass certain vertex subset and vertex partitioning problems for which the augmented vertex state set as described here is not finite. For instance, the problem Odd Neighborhood Cover, which has legal states of type ρ_i for any odd i and σ_j for any even j. We believe our approach can be applied to solve this problem on partial k-trees by defining vertex states based on parity of the number of selected neighbors.

For certain vertex partitioning problems, the algorithms given here may not be the appropriate view. In particular, the PARTITION INTO PERFECT MATCHINGS problem is defined in chapter 2.1 as a $minD_q$-problem, but we have no good bound on this parameter for which Theorem 6.4 would yield an algorithm polynomial in $|V(G)|$. Instead, we can derive an algorithm without utilizing the vertex state approach at all, bounding the number of separator states by $|I_k| = 3^k B(k)$, where $B(k)$ is the kth Bell number. Recall the definition of Partition into Perfect Matchings ([GT16] in [34]): For a graph G, find the minimum value of p for which there is a partition $V_1, V_2, ..., V_p$ of $V(G)$ such that $G[V_i], 1 \leq i \leq p$ has vertices of degree one only. We sketch a linear time algorithm for this problem on graphs of bounded treewidth (both [17] and [8] show only the existence of polynomial time algorithms). We observe that equivalent solutions to subproblems must induce identical partitions on the separator. We classify solutions by whether a given separator vertex has a mate among the other separator vertices, a mate among the reduced vertices, or no mate yet. Based on this we define a set of separator states and implementations of Initialize, Reduce, Join and Root-Optimize operations that give a linear time solution algorithm.

As mentioned in chapter 5.5 we intend to extend the class of vertex state problems beyond the vertex partitioning problems given here, hopefully resulting in better complexity bounds, or even the first polytime algorithms, for computing those parameters on partial k-trees.

One approach would enlarge the current definition of vertex partitioning problems to allow specification of certain restricted subgraphs between partition classes. This would encompass for, example, Broadcasting problems, using the following definition of these problems: a graph has broadcast time t or less if its vertices can be partitioned into classes $V_0, V_1, ..., V_t$ with the constraint that the original holders of the message to be broadcast are V_0, and that for any $i, 1 \leq i \leq t$ a matching covering V_i exists in the bipartite graph on vertices $\{v : v \in V_j \wedge j \leq i\}$ and edges $\{uv : u \in V_i \wedge v \in V_j \wedge j < i\}$.

Finally, we would like to extend the partial k-tree algorithm design methodology given in chapter 5 to the area of parallell partial k-tree algorithms.

Bibliography

[1] J. Abello, M.R. Fellows and J.C. Stillwell, On the complexity and combinatorics of covering finite complexes, *Australasian Journal of Combinatorics 4* (1991), 103-112;

[2] K.Abrahamson and M.Fellows, Finite automata, bounded treewidth and well-quasiordering, to appear in *Contemporary Mathematics* (1992).

[3] D. Angluin, Local and global properties in networks of processors, in *Proceedings of the 12th ACM Symposium on Theory of Computing* (1980), 82-93;

[4] D. Angluin and A. Gardner, Finite common coverings of pairs of regular graphs, *Journal of Combinatorial Theory B 30* (1981), 184-187;

[5] K.Appel and W.Haken, Every planar map is 4-colorable, *Bull. AMS 82* No. 5, 711-712 (1976)

[6] S.Arnborg, D.Corneil and A.Proskurowski, Complexity of finding embeddings in a k-tree, sl SIAM J. Alg. Disc. Meth. 8 , 277-284 (1987)

[7] S.Arnborg, S.Hedetniemi and A.Proskurowski (editors) *Algorithms on graphs with bounded treewidth*, Special issue of *Discrete Applied Mathematics*.

[8] S.Arnborg, J.Lagergren and D.Seese, Easy problems for tree-decomposable graphs, *J. of Algorithms 12(1991) 308-340.*

[9] S.Arnborg and A.Proskurowski, Linear-time algorithms for NP-hard problems on graphs embedded in k-trees, *TRITA-NA-8404, The Royal Institute of Technology* (1984).

[10] S. Arnborg and A. Proskurowski, Characterization and recognition of partial 3-trees, *SIAM J. Alg. and Discr. Methods 7 (1986) 305-314.*

[11] S.Arnborg and A.Proskurowski, Linear time algorithms for NP-hard problems on graphs embedded in k-trees, *Discr. Appl. Math. 23 (1989) 11-24.*

[12] D.W.Bange, A.E.Barkauskas and P.J.Slater, Efficient dominating sets in graphs, in: Ringeisen and Roberts, eds., *Applications of Discrete Mathematics*, SIAM, (1988).

[13] P.J.Bernhard, S.T.Hedetniemi and D.P.Jacobs, Efficient sets in graphs, *Discrete Applied Mathematics* 44 (1993).

[14] M.W.Bern, E.L.Lawler and A.L.Wong, Linear-time computation of optimal subgraphs of decomposable graphs, *J. of Algorithms 8(1987) 216-235.*

[15] N. Biggs, *Algebraic Graph Theory*, Cambridge University Press, 1974;

[16] H.L.Bodlaender, Dynamic programming on graphs with bounded treewidth, *Proceedings ICALP 88, LNCS vol.317 (1988) 105-119.*

[17] H.L. Bodlaender, A linear time algorithm for finding tree-decompositions of small treewidth, in *Proceedings STOC'93.*

[18] H.Bodlaender, M.Fellows and Hallett, Beyond NP-completeness for problems of bounded width: Hardness for the W-hierarchy, *Proceedings STOC 1994.*

[19] J.A.Bondy and U.S.R.Murty, *Graph theory with applications*, 1976.

[20] R.B.Borie, R.G.Parker and C.A.Tovey, Automatic generation of linear algorithms from predicate calculus descriptions of problems on recursive constructed graph families, *Algorithmica*, 7:555-582, 1992.

[21] J.A.Bondy and U.S.R.Murty, *Graph theory with applications*, 1976.

[22] K. Cameron, Induced Matchings, *Discrete Applied Mathematics* 24 (1989), 97-102.

[23] E.J.Cockayne, B.L.Hartnell, S.T.Hedetniemi and R.Laskar, Perfect domination in graphs, manuscript (1992), to appear in J. Combinatorics Inform. Systems Sci.

[24] E.J.Cockayne and S.T.Hedetniemi, Towards a theory of domination in graphs, *Networks*, 7 (1977), 211-219.

[25] S.Cook, The complexity of theorem-proving procedures, *Proc. 3rd STOC, ACM*, New York, (1971) 151-158.

[26] D.Corneil and J.Keil, A dynamic programming approach to the dominating set problem on k-trees. *SIAM Journal of Alg. Disc. Meth.* (1987), 8:535-543.

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

[28] B.Courcelle and M.Mosbah, Monadic second-order evaluations on tree-decomposable graphs, Proceedings of *17th Workshop on Graph-Theoretic Concepts in Computer Science*, (1991)

[29] M.E.Dyer and A.M.Frieze, Planar 3DM is NP-complete, *Journal of Algorithms*, vol.7, 174-184, (1983).

[30] J.Edmonds, Paths, trees and flowers, *Cand. Journal of Math.* 17 (1965) 449-467.

[31] A.Farley and A.Proskurowski, Computing the maximum order of an open irredundant set in a tree, Congr.Numer. 41 (1984) 219-228.

[32] M.Fellows and M.Hoover, Perfect domination, *Australian J. Combinatorics* 3 (1991), 141-150.

[33] J.F.Finck and M.S.Jacobson, On n-domination, n-dependence and forbidden subgraphs, in *Graph Theory with Applications to Algorithms and Computer Science*, Wiley (1984) 301-312.

[34] M.R. Garey and D.S. Johnson, *Computers and Intractability*, W.H.Freeman and Co., 1978;

[35] M. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*,

[36] D.Grinstead and P.Slater, A recurrence template for several parameters in series-parallel graphs, manuscript (1992).

[37] F.Harary, *Graph Theory*, 1969.

[38] S.T.Hedetniemi, Open problems in combinatorial optimization, manuscript (1994).

[39] S.T.Hedetniemi and R.Laskar, Recent results and open problems in domination theory, in: Ringeisen and Roberts, eds., *Applications of Discrete Mathematics* (SIAM, 1988).

[40] S.T.Hedetniemi and R.Laskar, Bibliography on domination in graphs and some basic definitions of domination parameters, *Discrete Mathematics* 86 (1990).

[41] S.M.Hedetniemi, S.Hedetniemi and T.Beyer, A linear algorithm for the Grundy (coloring) number of a tree, *Congressus Numerantium vol.* 36 (1982) pp.351-362

[42] P. Hell and J. Nešetřil, On the complexity of H-colouring, *Journal of Combinatorial Theory B 48* (1990), 92-110.

[43] I. Holyer, The NP-completeness of edge-coloring, *SIAM J. Computing 4* (1981), 718-720;

[44] M.S.Jacobson, K.Peters and D.R.Fall, On n-irredundance and n-domination, *Ars Combinatoria* 29B (1990) 151-160.

[45] R.Karp, Reducibility among combinatorial problems, in *Complexity of computer computations*, Plenum Press, New York, (1972) 85-103.

[46] J.Kratochvíl, Perfect codes in general graphs, monograph, Academia Praha (1991).

[47] J.Kratochvíl and P.Manuel, Generalized domination in chordal graphs, manuscript (1994).

[48] J.Kratochvíl, A.Proskurowski and J.A.Telle, Complexity of graph covering problems, to appear in *Proceedings of Workshop on Graph-Theoretic Concepts in Computer Science, Munchen 94*, LNCS 1994.

[49] D. König, Über graphen und ihre Andwendung auf Determinantentheorie und Mengenlehre, *Math. Ann. 77*, 1916, 453-465;

[50] J.Lagergren, private communication (1994).

[51] L.Lawson, J.Boland and T.Haynes, Domination from afar, presented at *25th SouthEastern Conference on Computing, Combinatorics and Graph Theory*

[52] F.T. Leighton, Finite common coverings of graphs, *Journal of Combinatorial Theory B 33* (1982), 231-238;

[53] J. van Leeuwen, Graph Algorithms, in *Handbook of Theoretical Computer Science vol. A* , Elsevier, Amsterdam, (1990) pg.550.

[54] J.Matoušek and R.Thomas, Algorithms finding tree-decompositions of graphs, *J. of Algorithms*,12 (1991) 1-22.

[55] A. McRae, PhD thesis, Clemson University (1994).

[56] Ø.Ore, Theory of graphs, *Amer.Math.Soc.Colloq.Publ. 38*, Providence (1962)

[57] J. Petersen, Die Theorie der regulären Graphen, *Acta Mathematica 15* (1891), 193-220;

[58] A.Proskurowski and M.Sysło, Efficient computations in tree-like graphs, in *Computing Suppl. 7,(1990) 1-15.*

[59] N. Robertson and P.D. Seymour, Graph minors II: algorithmic aspects of tree-width, *J. of Algorithms 7 (1986) 309-322.*

[60] D.Sanders, On linear recognition of tree-width at most four, manuscript (1992).

[61] P.Scheffler, Linear-time algorithms for NP-complete problems restricted to partial k-trees, R-MATH-03/87, Akademie der Wissenschaften der DDR (1987)

[62] K.Takamizawa, T.Nishizeki and N.Saito, Linear-time computability of combinatorial problvems on series-parallel graphs, *J. ACM 29(1982) 623-641.*

[63] J.A.Telle, Characterization of domination-type parameters in graphs, *Proceedings of 24th SouthEastern Conference on Combinatorics, Graph Theory and Computing, Congressus Numerantium vol. 94 (1993) 9-16.*

[64] J.A.Telle, Complexity of domination-type problems in graphs, to appear *Nordic Journal of Computing.*

[65] J.A.Telle and A.Proskurowski, Practical algorithms on partial k-trees with an application to domination-type problems, in *Proceedings Workshop on Algorithms and Data Structures, Montreal 93*, LNCS vol. 709 (1993) 610-621.

[66] J.A.Telle and A.Proskurowski, Efficient sets in partial k-trees, *Discrete Applied Mathematics* 44 (1993) 109-117.

[67] T.Wimer, Linear time algorithms on k-terminal graphs. Ph.D. thesis, Clemson University, South Carolina, (1988).