MATRIX PARTITIONS OF GRAPHS: ALGORITHMS AND COMPLEXITY

by

Mayssam Mohammadi Nevisi

M.Sc., Sharif University of Technology, 2009

B.Sc., Sharif University of Technology, 2006

A Thesis submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

in the
School of Computing Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Spring 2016

APPROVAL

Name: Mayssam Mohammadi Nevisi

Degree: Doctor of Philosophy

Title of Thesis: Matrix Partitions of Graphs: Algorithms and Complexity

Examining Committee: Dr. Binay Bhattacharya
Chair

Dr. Pavol Hell, Professor, Computing Science
Simon Fraser University
Senior Supervisor

Dr. Joseph G. Peters, Professor, Computing Science
Simon Fraser University
Supervisor

Dr. Ladislav Stacho, Professor, Mathematics
Simon Fraser University
Internal Examiner

Dr. Barnaby Martin, External Examiner,
Senior Lecturer,
Middlesex University, London

Date Approved: March 23, 2016

Abstract

Recently, there has been much interest in studying certain graph partitions that generalize graph colourings and homomorphisms. They are described by a pattern, usually viewed as a symmetric $\{0, 1, *\}$-matrix M. Existing results focus on recognition algorithms and characterization theorems for graphs that admit such M-partitions, or M-partitions in which vertices of the input graph G have lists of admissible parts. For (homomorphism) problems with costs, researchers have also investigated the approximability of the problem.

In this thesis, we study the complexity of these matrix partition problems. First, we investigate the complexity of counting M-partitions. The complexity of counting problems for graph colourings and graph homomorphisms has been previously classified, and most turned out to be $\#$ P-complete, with only trivial exceptions. By contrast, we exhibit many M-partition problems with interesting non-trivial counting algorithms; moreover these algorithms appear to depend on highly combinatorial tools. In fact, our tools are sufficient to classify the complexity of counting M-partitions for all matrices M of size less than four.

Then, we turn our attention to the homomorphism problems with costs. Previous results include partial classification of approximation complexity for doubly convex bipartite graphs. We complete these results and extend them to all digraphs. We prove that if H is a co-circular arc bigraph, then the minimum cost graph homomorphism problem to H admits a polynomial time constant ratio approximation algorithm. This solves a problem posed in an earlier paper. Our algorithm is obtained by derandomizing a two-phase randomized procedure.

In the final third of the thesis, we present a partial dichotomy for the complexity of exact minimization of homomorphism costs, when the cost function is a constant across the vertices of the input graph. We show that the dichotomy is complete when the target graph is a tree.

Keywords. partitions; homomorphisms; counting problems; approximation algorithms; polynomial algorithms; dichotomy

Acknowledgments

First of all, I would like to extend my utmost gratitude to my senior supervisor, Dr. Pavol Hell, who has supported me with his broad knowledge, guidance, and encouragements during my graduate studies at Simon Fraser University. I have learnt a lot from him, not only from his technical skills in computer science, but also from his way of thinking and his attitudes toward research.

I would like to thank my supervisor, Dr. Joseph G. Peters. I am grateful to Dr. Ladislav Stacho and Dr. Barnaby Martin who kindly accepted to be my examiners. I would also like to thank Dr. Binay Bhattacharya who accepted to be the chair of the examining committee. I am grateful to Dr. Arthur L. Liestman and Dr. Funda Ergun who helped me with valuable comments and discussions.

Part of the results presented in this thesis were joint work. I am grateful to my collaborators: Dr. Arash Rafiey, Dr. Monaldo Mastrolilli, and Dr. Miki Hermann.

Last but not least, I am grateful to my parents, Fatemeh and Ahmad, and my wife, Atefeh Mirsafian, for their unconditional support and strong motivations during my entire academic studies. Their support has been invaluable to me.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

In this chapter, we first briefly review the definitions required throughout the thesis in Section 1.1. Then, we summarize our contributions in Section 1.2 and explain the organization of the thesis in Section 1.3.

1.1 Definitions

In this section, we bring the definitions required in the rest of the thesis. This is accomplished in two parts. In the first part, we briefly review the main concepts of the computational complexity theory. In the second part, we continue with the definitions and terminology required in the context of graph theory for directed and undirected graphs.

1.1.1 Theory of Computation

We assume the reader is familiar with the standard computational model, particularly, with the definitions of *decision problems*, *deterministic* and *nondeterministic Turing machines*, the computational complexity classes P and NP, the concepts of *complete* problems for a class, and polynomial time many to one (Karp) reductions [90]. We refer the interested reader to the text book [5] for further reading.

A *function problem* is a problem that can be formulated as a function. In other words, a problem $\mathcal{P}$ is a function problem if there exists a function f such that the solution of every instance x of $\mathcal{P}$ can be represented by $f(x)$. For convenience, we assume that the instance is encoded as a binary string, or a tuple of binary strings, and also every feasible solution

is encoded as a binary string as well. Let $\mathcal{P}$ be a function problem and $f : \mathcal{U}^n \to \mathcal{U}$ be its corresponding function (where $\mathcal{U}$ is $\{0,1\}^*$, the set of all binary strings, and n is any positive integer). A Turing machine M *computes* f (or, *solves* $\mathcal{P}$) if it writes $f(x)$ on the output tape and halts whenever started with binary string x written on the input tape for every possible input x. Furthermore, M *runs* (i.e., computes f) in time $T : \mathbb{N} \to \mathbb{N}$ when, for every input x, M requires at most $T(|x|)$ computational steps to compute $f(x)$. If, for some function T', $T \in \mathcal{O}(T')$, we may also say that M runs in time $\mathcal{O}(T')$.

The analogue of the class P for function problems is the complexity class FP, the set of all function problems that can be solved efficiently, i.e., using deterministic Turing machines that run in polynomial time. Some function problems involve finding a solution that minimizes (or maximizes) a cost function $c(x) : \{0,1\}^* \to \mathbb{N}$ over all feasible solutions x. For example, consider the following function problem. Given an input graph G, find the maximum size of any independent set in G. In this case, for every feasible solution, which is an independent set in G, the cost function is the size of the independent set, and the goal is to maximize this cost function. These kind of problems are called *optimization problems*. For every optimization problem, there is a corresponding decision problem that takes an additional input parameter k, and asks whether there is a solution of cost k. For example, the decision version of the independent set problem can be formulated as follows. Given a simple graph G and an integer k, does G have an independent set of size (at least) k?

Let $\mathcal{P}$ be an optimization problem. An *approximation algorithm* for $\mathcal{P}$ is an algorithm that finds an approximate solution. A *c-approximation algorithm* is an approximation algorithm with a multiplicative guarantee c. That is, when $\mathcal{P}$ is a minimization problem, the algorithm finds a solution which is **at most** c times the optimum solution, and when $\mathcal{P}$ is a maximization problem, it finds a solution which is **at least** c times the optimum solution. (For minimization problems, $c > 1$ and for maximization problems, $c < 1$). In general, c can be expressed as a function of the input size, n.

A *constant-ratio* (or *constant-factor*) approximation algorithm is a c-approximation algorithm for some constant c. The set of optimization problems that admit a polynomial time constant-ratio approximation algorithm is denoted by APX. We say that $\mathcal{P}$ has a *polynomial time approximation scheme* when there is a constant-ratio $(1+\epsilon)$-approximation algorithm for every $\epsilon > 0$ (ϵ-approximation for maximization problems). The set of optimization problems that admit a polynomial time approximation scheme is denoted by PTAS. By definition, PTAS $\subseteq$ APX. Arora, Lund, Motwani, Sudan and Szegedy proved that unless

P = NP, there are problems in APX that do not admit a polynomial time approximation scheme [6]. In the rest of this thesis, we only discuss polynomial time approximation algorithms and when we say *approximation algorithm*, we always mean a polynomial time approximation algorithm. We say that a problem is *not approximable* when, unless P = NP, there is no polynomial time approximation algorithm with a multiplicative guarantee possible for it.

Some function problems involve finding the number of solutions of a specific problem. These kind of problems are called *counting problems*. A *counting Turing machine* is a special non-deterministic Turing machine that outputs the number of accepting paths, for every given input, on an additional output tape. Counting Turing machines were introduced by Valiant in 1979 [127]. He also defined the complexity class # P as the class of functions f that can be computed using counting Turing machines that run in polynomial time.

It turns out that for some of the NP-complete problems, the many to one (Karp) reductions do not maintain a tractable relation between the number of solutions of the problems (there is not a straightforward mapping between the exact number of solutions of the problems involved in the reduction). Denote by $X^{\mathcal{O}}$ the class of all function problems with Turing machines that belong to complexity class X and consult oracle $\mathcal{O}$. A problem $\mathcal{P}$ is # P-*hard* if for every problem $\mathcal{Q}$ in # P, there is a *counting reduction* from the problem $\mathcal{Q}$ to the problem $\mathcal{P}$, or equivalently, if $\#\,\mathrm{P} \subset \mathrm{FP}^{\mathcal{P}}$. Furthermore, $\mathcal{P}$ is # P-*complete* if $\mathcal{P}$ is # P-hard and $\mathcal{P} \in \#\,\mathrm{P}$ [127]. A counting reduction is very similar to the reductions between NP-complete problems. The only difference is that we should be able to keep track of the number of solutions. Formally, a counting reduction consists of two parts. The first part is a polynomial time computable function that maps an instance a of $\mathcal{Q}$ to an instance b of $\mathcal{P}$. The second part is another polynomial time computable function that maps the number of solutions of b to the number of solutions of a.

1.1.2 Graphs and Digraphs

An *undirected graph* G is an ordered pair of two sets, a set of *vertices* $V(G)$ and a set of *edges* $E(G)$, and every edge is associated with an unordered pair of two (not necessarily different) vertices.

Let G be an undirected graph, $e \in E(G)$ be an edge and $\{u, v\}$ be its associated pair of vertices. We call u and v the *end*s of the edge e and say that e *joins* u and v. Also, we say that u and v are *adjacent* in G, and e is *incident* with u and v. Moreover, we say that u is

a *neighbour* of v (and vice versa). For convenience, we will often use uv to implicitly refer to the edge joining u and v. The *open neighbourhood* (or simply, the *neighbourhood*) of a vertex $v \in V(G)$, denoted $N_G(v)$, is the set of all vertices u such that u and v are adjacent. The *closed neighbourhood* of v, denoted $N_G[v]$, is $N_G(v) \cup \{v\}$. The *degree* of a vertex v in G, denoted $d_G(v)$, is the size of its open neighbourhood.

In the rest of this thesis, we will often omit the word undirected and use *graph* for undirected graphs. We will use letters G and H to denote (undirected) graphs and letter D to denote digraphs (defined below). For simplicity, and when G can clearly be recognized from the context, we may use V and E instead of $V(G)$ and $E(G)$, respectively. Also, we may omit G and use $N(v)$, $N[v]$, and $d(v)$ to refer to the open and closed neighbourhoods, and the degree of vertex v, respectively. Similarly, we may use V, A, $N^-(v)$, $N^+(v)$, $d^-(v)$ and $d^+(v)$ when the digraph D can be identified clearly.

A *loop* is an edge that joins a vertex to itself. Two or more edges are *parallel* when they have the same pair of ends. A *simple graph* is a graph without loops and parallel edges. An *irreflexive graph* is a graph with no loops (but possibly with parallel edges). A *reflexive graph* is a graph in which every vertex has a loop. A graph is *finite* when its vertex set and edge set are both finite sets. In this thesis, we assume all graphs are finite and without parallel edges.

Graphs are usually represented by their *adjacency list* or by their *adjacency matrix*. In the adjacency list representation, there is a set for every vertex $v \in V(G)$ which contains the vertices that are adjacent to v, while in the adjacency matrix representation, a matrix A is used whose rows and columns are indexed by vertices of G, and A_{ij} gives the number of edges between vertex i and j.

A *subgraph* of a graph $G = (V, E)$ is a graph whose set of vertices and edges are subsets of V and E, respectively. Equivalently, H is a subgraph of G if it can be obtained from G by deleting some edges and/or some vertices (together with their incident edges) of G. An *induced* subgraph of G is a subgraph of G that can be obtained from G using only vertex deletions. Moreover, we denote by $G[S]$ the induced subgraph of G whose vertex set is S, by $G - v$ the induced subgraph $G[V - \{v\}]$, and by $G - S$ the induced subgraph $G[V - S]$.

A graph G is *connected* if for every partition of its vertices into two non-empty disjoint parts X and Y, there is at least one edge with one end in X and one end in Y. Otherwise, the graph is *disconnected*. A *component* (or *connected component*) of G is a maximal induced connected subgraph of G. Thus, a connected graph consists of one single connected

component. Furthermore, up to order, there is a unique representation of every graph by its connected components.

A *walk* is a sequence $W = v_0v_1 \cdots v_k$ of (not necessarily distinct) vertices, in which every two consecutive vertices in the sequence (v_i and v_{i+1}, $0 \leq i < k$) are adjacent. We call v_0 and v_k the *end*s of the walk and $v_1, \cdots, v_{k-1}$ the *internal vertices*, and say that W *connect*s v_0 to v_k, or W is a v_0v_k-walk. A walk is *closed* when its ends are identical, and is a *trail* when its edges are all distinct.

A *k-path* is a simple graph with k vertices $v_1, \cdots, v_k$ and $k-1$ edges $e_i = v_iv_{i+1}$ ($1 \leq i < k$). A *path* is a k-path for some $k \geq 1$. Equivalently, a path is a walk in which all vertices are distinct. A *k-cycle* ($k > 2$) is a simple graph with k vertices $\{v_0, v_1, \cdots, v_{k-1}\}$ and k edges $e_i = v_iv_{i+1}$ ($0 \leq i \leq k-1$, indices modulo k). A *1-cycle* consists of a single vertex and a loop, and a *2-cycle* consists of two vertices with two parallel edges joining them. A *cycle* is a k-cycle for some positive integer k. We refer to the number of edges in a path or cycle as its *length*. Hence, a path on $k+1$ vertices, denoted P_{k+1}, and a cycle on k vertices, denoted C_k, have length k. We say that a cycle or a path is *even* if it is of even length, and is *odd* otherwise.

A *bipartite graph* is a graph whose vertex set can be partitioned into two parts X and Y so that no edge has both ends in the same part. Hence, bipartite graphs are irreflexive by definition. We also call (X, Y) a *bipartition* of G and reserve the word *bigraph* (and the notation $G[X, Y]$) for a bipartite graph G with a given bipartition (X, Y). When the bipartition is given, we may refer to the vertices in part X as white vertices, and to the vertices in part Y as black vertices. It is well known that a graph is bipartite if and only if it does not contain any odd cycle (See [14]).

The definition of bipartite graphs is generalized as follows. For every $k \geq 2$, a *k-partite* graph is a graph whose vertex set can be partitioned into k parts so that no edge has both ends in the same part. Again, k-partite graphs are irreflexive by definition.

A *tree* is a simple connected graph which does not have any subgraph which is a cycle. There is always a unique path between any two vertices in a tree (as otherwise, the graph would be disconnected, or have a cycle as a subgraph). The vertices of a tree are sometimes called its *nodes*. A *leaf* in a tree is a node with degree one. A *rooted tree* is a tree with a designated vertex called its *root*. Let T be a tree with root r. For any vertex $v \in V(T)$, *level* of v is the length of the path from v to r. Let $p(v)$ denote the the immediate vertex after v in the path from v to r. We say that $p(v)$ is the *parent* (or *predecessor*) of v, and v is a *child*

(or *successor*) of $p(v)$. The vertices whose parents are the same vertex are called *siblings*. A *forest* is a simple graph in which every component is a tree. A *reflexive tree* is a graph in which every vertex has a loop, and removing all the loops gives a tree. A *partially reflexive tree* is a connected graph (with loops allowed) that does not contain any subgraph which is a k-cycle for every $k > 1$. A partially reflexive tree is *loop-connected* when the subgraph induced on vertices with loops is connected.

A *complete graph* is a simple graph in which every two distinct vertices are adjacent. A *reflexive complete graph* is a graph in which every vertex has a loop, and removing all the loops leaves a complete graph. The *clique covering number* of a graph G is the minimum number of cliques in G required to cover all of its vertices. A *complete bigraph* is a bipartite graph where every white vertex is adjacent to every black vertex. An *empty graph* is a graph with no edges. A complete subgraph is often called a *clique* (or k-*clique* when it has k vertices), and a complete bipartite subgraph is called a *biclique* (or (k, l)-*biclique* when it has k white vertices and l black vertices). An empty subgraph is called an *independent set* or a *stable set*. We let K_n stand for the complete graph on n vertices, K_n^{ref} for the complete reflexive graph on n vertices, and $K_{p,q}$ for the complete bigraph on p white vertices and q black vertices.

A *cut vertex* is a vertex v that its removal from the graph disconnects (eliminates all paths between) two vertices in G. Equivalently, v is a cut vertex if there are distinct vertices $x \neq y$ (in the same component of G) such that there is no path between them in $G - v$. A *cutset* is a set of vertices that disconnect G. Not all graphs have cut vertices or cutsets, for example, a cycle has no cut vertices, and a complete graph has no cutsets. A *clique cutset* is a cutset whose vertices induce a clique. Similarly, a *stable cutset* is a cutset whose vertices induce an independent set (stable set).

A *homogeneous set* of a graph G is a non-empty subset $S \subset V$ that shares the same neighbourhood outside S: every vertex in $V - S$ is either adjacent to all vertices in S, or is not adjacent to any of them. A *skew partition* of a graph G is a partition of its vertex set into two non-empty parts such that one part induces a disconnected subgraph of G and the other induces a disconnected subgraph in the complement of G.

Let $G = (V, E)$ be a simple graph. The *complement* of G, denoted $\overline{G}$, is defined on the same vertex set V as follows. Two distinct vertices are adjacent in $\overline{G}$ if and only if they are not adjacent in G. A component of $\overline{G}$ is often called a *co-component*.

When G is bipartite, we also define a bipartite form of complement graph. Formally, let

$G[X,Y] = (V,E)$ be a bipartite graph. The *bipartite complement* of G, denoted $\overline{G[X,Y]}$, is the bipartite graph with the same bipartition as G, in which a white vertex $x \in X$ is adjacent to a black vertex $y \in Y$ if and only if x and y are not adjacent in G. When there is no chance of ambiguity, we may use $\overline{G}$ to denote the bipartite complement of G.

Given a family $S = \{S_1, S_2, \cdots, S_n\}$ of sets, its *intersection graph* is the graph whose vertex set is S and edge set is all pairs of vertices (sets) that intersect. That is, S_i and S_j are adjacent if and only if $|S_i \cap S_j| > 0$. As every set intersects itself, intersection graphs are reflexive by definition.

An *interval graph* is an intersection graph of a set of intervals on the real line. That is, each vertex in an interval graph is represented by an interval, and two vertices are adjacent if and only if their corresponding intervals intersect. Lekkerkerker and Boland proved in [98] that a graph has an interval representation if and only if it does not contain as an induced subgraph any cycle of length at least four, or an *asteroidal triple*; a set of three distinct vertices such that there is path between any two of them that does not contain any neighbour of the third vertex. A *proper* interval graph is a graph that admits an inclusion-free interval representation.

It is well known that interval graphs and proper interval graphs can be recognized in linear time (see for example [16, 93] for interval graphs and [28, 33, 34] for proper interval graphs). Wegner studied proper interval graphs in his PhD thesis ([135]) and proved the following result.

Lemma 1.1. *[135] A graph has a proper interval representation if and only if it does not contain as an induced subgraph a cycle of length at least four, a claw, a net, or a tent.*

Claw, net and tent are depicted in Figure 1.1.

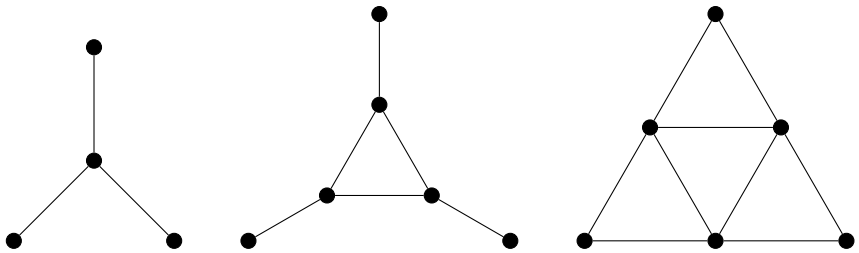

Figure 1.1: The claw, net and tent (drawn without loops)

A bipartite graph $G[X, Y]$ is an *interval bigraph* when there is an interval representation of vertices such that $x \in X$ and $y \in Y$ are adjacent if and only if their corresponding intervals intersect. Interval bigraphs can be recognized in polynomial time [110]. An interval bigraph is *proper* if it admits an interval representation such that the set of intervals representing its white vertices and the set of intervals representing its black vertices are both inclusion-free. Spinrad, Brandstädt and Stewart showed that proper interval bigraphs can be recognized in linear time [119].

A graph G is a *circular arc graph* if it is the intersection graph of a family of arcs on a circle; every vertex is represented by an arc on the circle, and two vertices are adjacent if and only if their corresponding arcs intersect. Circular arc graphs are a natural generalization of interval graphs, going from the intervals on the line to the arcs on the circle. In fact, an interval representation can be obtained from a circular arc representation when at least one point on the circle is not covered by any arc, by cutting the circle at that point. Circular arc graphs can be recognized in linear time (see [107]). Recently, Francis, Hell and Stacho [54] presented a forbidden structure characterization of circular arc graphs and developed the first polynomial time certifying algorithm for recognition of circular arc graphs (that outputs an obstruction subgraph in case the input does not admit a circular arc representation). For a recent survey on circular arc graphs see [35].

A circular arc graph is *proper* if it admits a circular arc representation in which no arc contains another. Tucker [125] characterized proper circular arc graphs using an infinite family of minimal induced forbidden subgraphs. Later, Deng, Hell and Huang developed linear time algorithms for recognition of proper circular arc graphs [34].

A graph G is a *subtree* graph if it is an intersection graph of a family of subtrees of a tree T; every vertex of G is represented by a subtree of T and two vertices are adjacent if and only if their corresponding subtrees intersect (share a vertex).

A *chordal* graph is a graph in which every cycle of length at least four has a *chord*, an edge that joins two non-adjacent vertices in the cycle. Equivalently, a graph is chordal if and only if it does not have any induced cycle of length greater than three. Gavril [57] proved that a graph is chordal if and only if it has a subtree representation. A vertex v is called a *simplicial vertex* if the induced subgraph $G[N[v]]$ is a clique. An ordering $v_1, v_2, \cdots, v_n$ of vertices of G is a *perfect elimination ordering* if v_i is a simplicial vertex in $G[\{v_i, v_{i+1}, v_n\}]$. It is also known that a graph is chordal if and only if it admits a perfect elimination ordering. Chordal graphs can be recognized in linear time [117, 122, 123].

A *split graph* is a graph whose vertices can be partitioned into two sets, an independent set and a clique. It is easy to see that split graphs cannot contain any induced cycle of length at least four, and hence they are chordal.

Bang-Jensen and Hell refined Wegner's characterization [9]. For graphs G and H, they said that G is a *multiple* of H when it can be obtained from H by replacing each vertex $u \in V(H)$ by a complete graph K^u and joining vertices of K^u and K^v whenever u and v are adjacent in H. They showed that a chordal graph G that is both claw-free and net-free but contains a tent as an induced subgraph is indeed a multiple of tent. Thus, they deduced the following characterization of proper interval and proper circular arc graphs.

Corollary 1.1. *[9] A chordal graph G is a proper interval graph if and only if it is claw-free and net-free and is not a multiple of the tent.*

Corollary 1.2. *[9] A chordal graph G is a proper circular arc graph if and only if it is claw-free and net-free.*

A *colouring* (also, *vertex-colouring*) of a graph G is a mapping $f : V(G) \to \mathbb{N}$, i.e., an assignment of *colours* (represented by positive integers) to the vertices of G. A *proper colouring* is a colouring in which all adjacent vertices map to different colours. We usually omit the word *proper* for simplicity as we only discuss proper colourings. A *k-colouring* is a colouring that uses at most k different colours. A graph is *k-colourable* if it admits a k-colouring. The *chromatic number* of a graph G is the smallest integer k such that G is k-colourable.

The *colouring problem* for graphs is the decision problem that asks, for a simple input graph G and an integer k, whether G admits a k-colouring. Karp proved that this problem is NP-complete [90]. For every positive integer k, the *k-colouring problem*, denoted *k-COL*, is the decision problem that asks whether an input graph G is k-colourable. The k colouring problem is NP-complete in general unless $k = 1, 2$ [56].

A *directed graph*, also called *digraph*, is an ordered pair $(V(G), A(G))$ where $V(G)$ is the set of vertices, and $A(G)$ is the set of *arcs*, or directed edges, disjoint from $V(G)$. While every edge in an undirected graph is an unordered pair, the vertices associated with an arc in a directed graph are ordered, hence defining a *direction* on the edge from one vertex to the other. For any digraph D, we can define an undirected graph by removing directions from its arcs. This is called the *underlying* graph of D and is denoted by $G(D)$. It is also possible to construct a directed graph D based on any undirected graph G by assigning an

arbitrary direction to every edge. This is called an *orientation* of G. Similarly, an *oriented walk*, an *oriented path* and an *oriented cycle* are a walk, path and a cycle with edges oriented in arbitrary directions.

Let D be a directed graph, $e \in E(G)$ be an arc and (u, v) be its associated pair of vertices. We say that u *dominates* v, and v is dominated by u. We call u the *head* of e and v the *tail* of it. Also, we say that v is an *out-neighbour* of u and u is an *in-neighbour* of v. We denote by $N_D^-(v)$ the set of all in-neighbours of v, and by $N_D^+(v)$ the set of all out-neighbours of v. The *in-degree* (similarly, *out-degree*) of v is the size of $N_D^-(v)$ (respectively, $N_D^+(v)$) and is denoted by $d_D^-(v)$ (respectively $d_D^+(v)$).

Many concepts primarily defined for graphs can be extended to digraphs as well. In particular, a *loop* is an arc with the same head and tail, and two arcs are *parallel* if they join the same head to the same tail. A digraph is reflexive, irreflexive, or finite when its underlying graph is reflexive, irreflexive or finite, respectively. A *directed path* is a directed graph whose underlying graph is a path with all arcs oriented in the same direction. A *directed cycle* is a directed graph whose underlying graph is a cycle, and all arcs are oriented in the same direction.

A digraph D is *semicomplete* if for every pair of vertices $x, y \in V(D)$, x dominates y, or y dominates x (or both).

1.2 Our Contribution

We study certain graph partition problems that generalize graph colourings and graph homomorphisms. These partition problems require certain parts to be cliques or independent sets. They may also require two parts to be fully adjacent, or have no edges joining them. These constraints are usually encoded by a symmetric $\{0, 1, *\}$-matrix M. Diagonal values define the constraints on the parts: $M_{i,i} = 1$ means part i must be a clique and $M_{i,i} = 0$ means it must be an independent set, while $M_{i,i} = *$ does not signify any restrictions. Off-diagonal values define the constraints between parts. Again, $M_{i,j} = 1$ means that every vertex in part i must be adjacent to every vertex in part j, $M_{i,j} = 0$ means there are no edges between parts i and j, and $M_{i,j} = *$ means there is no restriction on the edges between those two parts.

When M has no 1s, the problem corresponds to a graph homomorphism problem. When all diagonal values of M are 0, and all off-diagonal values are $*$, then it corresponds to a

graph colouring problem. For homomorphism problems, there are optimization problems with respect to minimizing costs.

In this thesis, we study three graph partition problems closely connected under the umbrella notion of matrix partitions. For (homomorphism) problems with costs, researchers have also investigated the approximability of the problem.

- First, we investigate the complexity of counting M-partitions [69]. The complexity of counting problems for graph colourings and graph homomorphisms (special cases of M-partitions) have been previously classified, and most turned out to be $\#$P-complete, with only trivial exceptions where the counting problems are easily solvable in polynomial time [128, 38]. By contrast, we exhibit many other M-partition problems with interesting non-trivial counting algorithms; moreover, these algorithms appear to depend on highly combinatorial tools. In fact, our tools are sufficient to classify the complexity of counting M-partitions for all matrices M of size less than four. In the process of determining the complexity of counting matrix partition problems:

  - We present a novel polynomial time algorithm for counting the number of bisplit partitions of any input graph. A bisplit partition is a partition of a graph into an independent set, and a biclique.
  - We present algorithms for counting (non-empty) modules of graphs, or counting modules of graphs with special properties. By special properties, we mean requiring the module itself, its set of neighbours, or its set of non-neighbours to be an independent set, or a clique. Modules of graphs are explained in Section 3.2.

  It turns out that, among matrices not accounted for by the existing results on counting homomorphisms, all matrices which do not contain the matrices for independent sets or cliques yield tractable counting problems. Motivated by our results, Gobel, Goldberg, McQuillan, Richerby and Yamakami investigated one variant of this problem where every vertex of the input graph is equipped with a list of admissible parts [58] and proved that there is a dichotomy classification for this variant. Later, Dyer, Goldberg and Richerby extended our results for small matrices to matrices of size 4 [37]. We discuss these two works as well.

- Then, we turn our attention to the homomorphism problems with costs. Previous results include a 2-approximation algorithm for the so-called doubly convex bipartite

graphs, as well as hardness of approximation when the corresponding list homomorphism problem is NP-complete [105]. We extend these results by proving that the converse of the latter is also true, thus, showing that there is a dichotomy classification for the problem of approximately finding the minimum cost graph homomorphisms to any target (directed) graph H. In the process, we investigate the bottleneck version of the homomorphism problem with costs, and provide a complete dichotomy classification for it. The dichotomy coincides with the well-known dichotomy of list homomorphism problems for graphs. We then present a polynomial time approximation algorithm that requires oracle access to an algorithm for the corresponding bottleneck minimum cost graph homomorphism problem. The approximation ratio guaranteed by our oracle algorithm is n, the number of vertices in the input graph. In search for better approximation ratios, we focus on simple target graphs with tractable list homomorphism problems. We prove that if H is a co-circular arc bigraph, then the minimum cost graph homomorphism problem to H admits a polynomial time constant ratio approximation algorithm. This solves a problem posed in an earlier paper [105]. Our algorithm is obtained by derandomizing a two-phase randomized procedure. We show a similar result for graphs H in which all vertices have loops: if H is an interval graph, then the minimum cost homomorphism problem to H admits a polynomial time constant ratio approximation algorithm.

- Minimum cost graph homomorphism problems have been viewed as generalizations of list homomorphism problems. They also generalize two well-known graph colouring problems: the minimum colour sum problem (MCS) [95], and the optimum cost chromatic partition problem (OCCP) [94]. In both of these problems, the cost function meets a specific constraint: the cost of using a specific colour is the same for every vertex of the input graph. We study the minimum cost graph homomorphism problems with cost functions having the same property. It is not hard to see that when the original problem is polynomial time solvable, the problem with constrained costs is also polynomial. It turns out that, in most cases, when the original problem is NP-complete, the problem with constrained costs is also NP-complete. Specifically, we prove that this is case for trees and the dichotomy of the minimum cost graph homomorphism problems to trees coincides with the dichotomy of the problem with constrained costs. We also prove that when H is not chordal bipartite, then the

minimum constrained cost graph homomorphism problem is NP-complete.

1.3 Organization of the thesis

The rest of this thesis is organized as follows.

Chapter 2 is dedicated to preliminaries. We survey a wide range of related problems, and present existing results. We start by looking into graph homomorphism problems in Section 2.1. Then, we turn our attention to variants of homomorphism problems, including list homomorphism problems, surjective homomorphism problems, and digraph homomorphism problems in the following three sections. Then, we discuss problems with costs. We start with colouring problems that involve costs in Section 2.5. This consists of the minimum colour sum problem and the optimum cost chromatic partition problem. Then, we review the background on their generalization to graph homomorphisms in Section 2.6. Finally, we discuss the matrix partition problems and counting graph homomorphism problems in the last two sections.

In Chapter 3, we study the complexity of counting matrix partition problems. This is accomplished in three parts. First, we present decomposition techniques that we use to present polynomial counting algorithms for some matrix partition problems. These techniques fall into two categories. In the first category, our counting algorithms are based on the so-called sparse-dense partitions first introduced by Feder and Hell [50] in studying list matrix partition problems. For matrices in the second group, we present efficient counting algorithms by extending the well-known result of Gallai [55] on modular decomposition of a graph. These are accomplished in Section 3.2. Our last polynomial counting algorithm is for the matrix M that corresponds to counting bi-split partitions (partitions of a graph into an independent set, and a complete biclique) and is presented in Section 3.3. Finally, we provide a dichotomy classification for small matrices in Section 3.4. We also discuss two recent developments in studying counting matrix partitions that are published very recently. In particular, the dichotomy for list matrix partition problems by Gobel, Goldberg, McQuillan, Richerby and Yamakami, and a computer assisted proof that extends our results to matrices of size 4.

In the following two chapters, our focus is on matrix partition problems that correspond to graph homomorphism problems. In particular, we investigate the minimum cost graph homomorphism problems in Chapter 4. We start by studying the bottleneck version of

the problem in Section 4.2 and presenting a polynomial time approximation algorithm for all (directed) graphs. In Section 4.3, we bring the polynomial algorithm for MinHOM(H) first discovered by Gutin, Hell, Rafiey, and Yeo [62] and discuss an equivalent integer linear program for it. Finally, we present our constant ratio approximation algorithms for co-circular arc bigraphs and reflexive interval graphs in Sections 4.4 and 4.5, respectively.

In Chapter 5, we introduce the minimum constrained cost graph homomorphism problem. First, we argue in Section 5.2 that the weighted version of the problem (with polynomially bounded vertex weight) is equivalent to the problem without weights. Then, we prove in the following two sections that the problem with constrained costs remains NP-complete when H is not a chordal bipartite graph. Finally, we show a complete dichotomy for trees in Section 5.5.

In Chapter 6, we present a summary of the thesis and discuss some open problems and possible directions for future work.

Chapter 2

Background

This chapter is dedicated to presenting the background of matrix partition problems. We survey problems related to M-partition problems and present known results. These problems are organized as follows. First, we consider matrix partition problems for matrices that represent graphs, i.e., graph homomorphism problems. This includes the graph homomorphism problem in Section 2.1, list homomorphism and retraction problems in Section 2.2, surjective homomorphism and compaction problems in Section 2.3, and the homomorphism problem for digraphs in Section 2.4. Then, we investigate matrix partition problems that involve costs of mappings, or costs of colouring. To be specific, Section 2.5 is dedicated to the minimum colour sum and optimum cost chromatic partition problems, and Section 2.6 is devoted to the minimum cost graph homomorphism problem. Then, we turn our attention to general matrix partition problems (with and without lists), where matrices are taken over $\{0, 1, *\}$ in Section 2.7. Finally, we explore the corresponding counting problems in the last three Sections. In particular, we review the counting problems for matrix partitions that represent graphs, i.e., counting graph homomorphism and counting graph list homomorphism problems in Section 2.8, and discuss counting matrix partitions of graphs (both with and without lists) in Section 3.5.

2.1 Graph Homomorphisms

Let G and H be two undirected graphs. We say that a mapping $f : V(G) \to V(H)$ is a *homomorphism* if it preserves the edges, that is, for every $uv \in E(G)$, $f(u)f(v) \in V(H)$. When f is a homomorphism of G to H, we say that H is the target (of the homomorphism).

For a fixed graph H, the *homomorphism problem* for H, denoted *HOM(H)*, asks whether an input graph G admits a homomorphism to H. Homomorphism problems generalize graph colouring problems. In fact, the k-colouring problem is exactly the homomorphism problem to the complete graph on k vertices. Hence, the homomorphism problem is also often called the *H-colouring* problem. The same problem can be formulated for digraphs G and H, and, in fact, for the more general relational structures [53].

We have already mentioned in Section 1.1 that the k-colouring problem is NP-complete for every $k \geq 3$. As a consequence, the graph homomorphism problem is NP-complete whenever H is the complete graph K_k (for any $k > 2$). On the other hand, a number of tractable cases are easily identified. In particular, when H is an empty graph or has a loop, the homomorphism problem is trivial. This is because, when H is empty, the answer is **yes** if and only if G is empty, and when H has a loop, the answer is always **yes**, as all vertices of the input graph G can be mapped to a vertex in H that has a loop. So, we assume that H is a simple graph with at least one edge. It is not hard to see that when H is a bipartite graph, then the homomorphism problem can be solved in linear time [106]. The reason is that if G is not bipartite, then the answer is always **no**, and if G is bipartite, then the answer is always **yes** as we can choose any edge in H and map all vertices of G to its ends. Interestingly, these are the only tractable cases as Hell and Nešetřil proved that all the remaining cases are NP-complete.

Theorem 2.1. *[75] If H is bipartite then the H-colouring problem is in* P. *If H is not bipartite then the H-colouring problem is* NP*-complete.*

Maurer, Sudborough and Welzl conjectured Theorem 2.1 (also known as the H-colouring dichotomy) in 1981. They proved the NP-completeness of the problem in the special case that H is an odd cycle. Feder and Vardi [53] conjectured a general dichotomy for all relational systems. Bulatov, Jeavons and Krokhin [19] developed an algebraic machinery to study the problem and formulated Feder and Vardi's conjecture in algebraic terms. Bulatov revisited the above graph homomorphism dichotomy theorem fifteen years later and reproved it using general methods for studying constraint satisfaction problems [21]. Recently, Siggers has presented another proof of the H-colouring dichotomy [118]. His proof is shorter than the original proof of Hell and Nešetřil and does not need the algebraic machinery used in Bulatov's proof.

Recall that a bigraph is a bipartite graph with a fixed partition of vertices into white

and black vertices. We say that a homomorphism f is *colour-preserving* if it preserves the vertex colours of G, that is, f maps white vertices of G only to white vertices of H and black vertices of G only to black vertices of H.

2.2 List Homomorphisms and Retractions

The classic application of graph colouring is the problem of *scheduling*; we are given a set of *tasks* and a positive integer k, and want to know whether all the tasks can be scheduled to be executed in k time slots. Two tasks may interfere, i.e., they cannot be executed at the same time, probably because they need the same resource. This problem can be modelled as a *conflict graph*, a graph whose vertex set is the set of tasks with two vertices adjacent if and only if their corresponding tasks interfere. Clearly, every scheduling of the tasks gives a proper k-colouring of the conflict graph; we can use colour i for a vertex when its corresponding task is scheduled to run in time slot i. Conversely, every k-colouring gives a feasible scheduling of the tasks, as the tasks corresponding to every vertex coloured i are independent, and thus, they can be executed in the same time slot i. Hence, the scheduling problem defined above is exactly the colouring problem.

Biro, Hujter and Tuza, motivated by applications in scheduling and VLSI theory, introduced a generalization of the k-colouring problem called *precolouring extension* and denoted *PrExt* [12]. In this modified version, part of the input graph G is properly coloured (with at most k colours) and the question is whether this prescribed colouring can be extended to a proper k-colouring for the whole graph G. In a series of papers, the same authors studied this problem and its variants for some important classes of graphs including interval graphs, some classes related to bipartite graphs, and some classes of perfect graphs [12, 81, 82]. Since then, the precolouring extension problem has attracted many researchers and has been studied extensively [3, 2, 104, 103, 133, 134, 115, 40].

A further generalization is to restrict the set of admissible colours at each vertex as follows. Together with the graph $G = (V, E)$, there are lists $L(v) \subset \{1, 2, \cdots, k\}$ for every vertex $v \in V$ as part of the input, and the question is whether there exists a k-colouring of G with respect to the lists L. This is called the *list colouring problem*. Indeed, list colouring was introduced earlier than the precolouring extension by both Vizing [132] and nearly simultaneously by Erdos, Rubin and Taylor [41] and has been studied extensively for decades (see, for example, the surveys [126, 15], or the book [87]).

Similarly, graph homomorphism problems have also been considered with local constraints. Let G and H be two graphs. A natural and practical approach to the graph homomorphism problem is, exactly similar to the generalization of the k-colouring problem with lists, introducing lists of admissible vertices of the target graph H for every vertex of the input graph G as follows. Given graphs H and G, and lists $L(v) \subset V(H)$ for all vertices $v \in V(G)$, a *list homomorphism* of G to H with respect to the lists L, is a homomorphism $f : V(G) \to V(H)$ such that $f(v) \in L(v)$ for all $v \in V(G)$. For a fixed graph H, the *list homomorphism problem* for H, denoted *LHOM-H*, asks whether an input graph G together with lists L admits a list homomorphism to H. The list homomorphism problem was introduced by Feder and Hell in [43]. It generalizes graph homomorphism, list colouring, and precolouring extension problems.

The complexity of the list homomorphism problem for graphs has already been classified. For reflexive graphs H, Feder and Hell [43] proved that the problem is NP-complete when H contains a chordless cycle (of length at least 4), or an *asteroidal triple*. Recall from Section 1.1 that a (reflexive) graph is an interval graph if and only if it does not contain any chordless cycle or an asteroidal triple. Hence, they concluded that the list homomorphism problem for reflexive graphs is NP-complete when H is not an interval graph. They also provided a polynomial time reduction of the problem for interval graphs H to the 2-satisfiability problem which is known to be solvable efficiently [56].

Theorem 2.2. *[43] Let H be a reflexive graph. If H is an interval graph, then LHOM-H is polynomial time solvable. If H is not an interval graph, then LHOM-H is* NP*-complete.*

The H-colouring dichotomy (Theorem 2.1) implies that, in the case of irreflexive graphs, LHOM-H is NP-complete when H is not bipartite, i.e., contains an odd cycle. Feder, Hell and Huang [46] proved that there are also other structures that make the list homomorphism problem intractable, including large even cycles (with size at least 6), and *special edge-asteroids*. An edge-asteroid in a bipartite graph $G = [X, Y]$ is a set of $2k+1$ edges u_iv_i and $2k+1$ paths P_i $(0 \leq i \leq 2k+1,\ k > 0,\ u_i \in X,\ v_i \in Y)$ that meet the following criteria for every $0 \leq i \leq 2k+1$.

- Each path P_i joins u_i to u_{i+1}
- $V(P_i) \cup \{v_i, v_{i+1}\}$ does not contain any neighbours of u_{i+k+1} or v_{i+k+1}

where all indices are module $2k+1$. An edge-asteroid is *special* if it has no edges between

the ends of the first edge (u_0v_0) and $V(P_i) \cup v_i$ for every $0 < i \leq 2k+1$. The authors then took advantage of the forbidden structure for the graphs that are the complement of circular arc graphs by Trotter and Moore [89] and concluded the following classification for irreflexive graphs.

Theorem 2.3. *[46] Let H be a graph (without loops). The list homomorphism problem to H is polynomial time solvable if the complement of H is a circular arc graph of clique covering number two, and is* NP*-complete otherwise.*

A graph has clique covering number two if and only if its complement is a bipartite graph, hence, the term *co-circular arc bigraph* has been equivalently used to refer to the class of graphs whose complement is a circular arc graph with clique covering number two, i.e., graphs H for which LHOM-H is tractable. Despite the apparent differences in their definition, co-circular arc graphs and interval graphs exhibit certain natural similarities (See [43, 46] for example).

Feder, Hell and Huang studied the list homomorphism problem for graphs with loops allowed. They introduced a new geometric representation called *bi-arc representation* [47] that generalized both interval graph representation and co-circular arc representation. Let C be a circle with two fixed points p and q. A *bi-arc* is an ordered pair of arcs (N, S) on C such that N contains p but not q, and S contains q but not p. The points p and q are often referred to as the *north pole* and *south pole*, respectively. A graph is a *bi-arc graph* if vertices of H can be represented by bi-arcs such that for any two vertices x, y, represented by $(N_x, S_x), (N_y, S_y)$ respectively, N_x and S_y intersect if and only if x and y are adjacent (same for N_y and S_x). It turns out that the complexity of the list homomorphism problem for general graphs (in which loops are allowed) is closely related to the bi-arc graphs, as stated in Theorem 2.4.

Theorem 2.4. *[47] When H is any bi-arc graph, LHOM-H is polynomial time solvable. Otherwise LHOM-H is* NP*-complete.*

We explain the polynomial algorithm as in [47] (see also [53]), below. First, remove from $L(u)$ any vertex $a \in V(H)$ that does not have a loop, for all vertices $u \in V(G)$ that have a loop. Then, for each pair $u, v \in V(G)$, define $S_{uv} = \{L(u) \times L(v)\} \cap E(H)$ when $uv \in E(G)$, and $S_{uv} = \{L(u) \times L(v)\}$ otherwise. Notice that if there exists a homomorphism f of G to H with respect to the lists L, then we must have $f(u), f(v) \in S_{uv}$. Then, for every three

distinct vertices $u, v, w \in V(G)$ perform the following update operation until there are no more updates possible or one of the sets becomes empty.

$$S_{uv} \leftarrow \{ab \in S_{uv} : \exists c \in V(H), ac \in S_{uw} \text{ and } bc \in S_{vw}\}$$

Obviously, $f(u)f(v) \in S_{uv}$ holds after every update operation. Thus, if any set becomes empty, then there is no homomorphism of G to H with respect to the lists L and the answer is **no**. Interestingly, the authors in [47] showed that the converse is also true for bi-arc graphs, and if the algorithm stops with all lists being non-empty, then the answer is **yes**: start with an arbitrary set of two vertices $U = \{u, v\}$, choose any pair $(a, b) \in S_{uv}$, and set $f(u) = a$ and $f(v) = b$. Then start adding vertices w of $V(G) - U$ to U one by one. At each step, select a vertex c such that $f(u)c \in S_{uw}$ for all $u \in U$ and set $f(w) = c$.

Bi-arc graphs contain both interval graphs and co-circular arc bigraphs. Furthermore, interval graphs are the only reflexive graphs with bi-arc representation, and co-circular arc bigraphs are the only irreflexive graphs with bi-arc representation [47]. Hence, the bi-arc representation seems to be a natural generalization of both the interval representation and the co-circular arc bigraph representation.

In addition to the original version mentioned above, researchers have shown interest in variants of the list homomorphism problem, where the input is required to meet certain conditions.

Let H be a fixed graph.

The *connected list homomorphism problem* to H, is the problem that, given a graph G together with lists L such that each list $L(v)$ induces a connected subgraph of H, asks whether there is a list homomorphism of G to H. In the case of reflexive graphs, the authors in [43] proved that this problem is polynomial time solvable when H is a chordal graph, and is NP-complete otherwise.

In the *complete list homomorphism problem* to H, the input is a graph G and lists L such that each list $L(v)$ induces a complete subgraph of H, and the question is, again, whether there is a homomorphism of G to H with respect to the lists L. When H is an irreflexive graph, the authors in [46] proved that the problem is polynomial time solvable when H is a triangle free graph, and is NP-complete otherwise.

The *one-or-all list homomorphism problem* is the problem that, for an input graph G and lists L such that each list $L(v)$ is either a single vertex of H or the entire set $V(H)$,

asks if there is a list homomorphism of G to H. One-or-all list homomorphism problems generalize precolouring extension problems (with H being a complete graph). Feder and Hell proved that the problem is NP-complete when H is a reflexive cycle of length $k \geq 4$ [43].

Another interesting related problem is the *graph retraction problem*. Let G be a graph and H be an induced subgraph of G. A homomorphism r of G to H is a *retraction* if for every vertex $v \in V(H)$, $r(v) = v$. When there exists a retraction of G to H, we say that H is a *retract* of G and G *retracts* to H. Let H be a fixed graph. The *retraction problem* for H, denoted RET-H, asks whether an input graph G, that contains H as an induced subgraph, retracts to H. Hell studied graph retractions in his Ph.D. thesis in 1972 [68]. Since then, retraction problems have gained noticeable amount of interest from the researchers [113, 28, 111, 8, 112, 7, 91, 130, 129, 131, 18, 101, 92].

The complexity of the retraction problems is an open problem in general, but several special cases have been classified. When H is a reflexive k-cycle ($k \geq 4$), the retraction problem is known to be NP-complete, and when H is a reflexive chordal graph, RET-H is polynomial time solvable [43]. In the case of irreflexive graphs, Bandelt, Dahlmann and Schutte proved that the problem is polynomial time solvable when H is a chordal bipartite graph [8]. Feder, Hell and Huang proved that RET-H is NP-complete when H is an even cycle of length at least 6 [46]. Vikas classified the retraction problem to all graphs with at most four vertices [131].

Feder and Hell proved in [43] that, for every fixed graph H, the retraction problem for H and the one-or-all list homomorphism problem to H are polynomially equivalent.

2.3 Surjective Homomorphisms and Compactions

Other related problems are the *vertex-surjective homomorphism* and the *edge-surjective homomorphism* problems. Let G and H be two graphs. A homomorphism h of G to H is *vertex-surjective* if for every vertex a of the target graph H, there is at least one vertex u of G such that $h(u) = a$. Moreover, if h is a vertex-surjective homomorphism and for every edge $ab \in E(H)$ $(a \neq b)$ there is an edge $uv \in E(G)$ such that $h(u) = a$ and $h(v) = b$, then h is an *edge-surjective* homomorphism.

For a fixed graph H, the *surjective homomorphism problem*, also called as the *Surjective H-Colouring problem*, asks whether an input graph G admits a vertex-surjective homomorphism to H. Analogously, the *edge-surjective homomorphism problem*, also often called

the *compaction problem* and denoted *COMP-H*, asks whether an input graph G admits an edge-surjective homomorphism to H. For historical reasons, in the compaction problem, the loops are exempt from the surjectivity constraint, i.e., the required homomorphism is surjective on all edges but not necessarily all loops.

The complexity of surjective homomorphism problems, and of compaction problems has not been completely classified (see [13]). Chen [26] has extended some of the algebraic machineray developed for the homomorphism problems to surjective homomorphisms. Golovach, Paulusma and Song proved a dichotomy when H is a partially reflexive tree [59] as follows.

Theorem 2.5. *[59] For any fixed tree H, the Surjective H-Colouring problem is polynomial time solvable if H is loop-connected, and* NP*-complete otherwise.*

Recently, Martin and Paulusma proved that when H is the reflexive 4-cycle, the surjective homomorphism problem to H is NP-complete [102].

There is a close relation between compaction and retraction problems. In particular, it is known that, for every fixed graph H, COMP-H is polynomial time solvable when the corresponding retraction problem, RET-H, is polynomial time solvable [129]. But it is not known if the converse is true. Vikas showed that for every fixed reflexive (or bipartite) graph H, there exists a reflexive (resp. bipartite) graph H' so that the problems RET-H and COMP-H' are polynomially equivalent [129].

2.4 Digraph Homomorphisms

The definition of graph homomorphism can easily be extended to digraphs. Let D and H be digraphs. A mapping $h : V(D) \to V(H)$ is a *homomorphism* of D to H if it preserves the arcs, that is, h is a homomorphism if for every arc $uv \in A(G)$ we have $f(u)f(v) \in A(H)$. Similarly, we can define list homomorphisms for digraphs when there is a list of admissible vertices of H for every vertex of D. The homomorphism and list homomorphism problems for digraphs are defined analogously.

The complexity of the digraph homomorphism problem is an open problem and only special cases have been settled. In particular, Feder studied the problem when the target digraph H is an oriented cycle [42].

In contrast to the digraph homomorphism problem, the complexity of the list homomorphism problem for directed graphs has been classified. Recall that in the case of undirected

graphs, existence of certain structures in the target graph makes the problem NP-complete. In the case of directed graphs, Hell and Rafiey proved that the complexity of the list homomorphism problem depends on existence of a fairly complicated structure that they called *directed asteroidal triple* defined below.

Theorem 2.6. *[79] Let H be a digraph.*
If H contains a DAT, then LHOM-H is NP*-complete.*
If H is DAT-free, then LHOM-H is polynomial time solvable.

The dichotomy for list homomorphisms was later extended by Bulatov [20] to all constraint satisfaction problems.

Let H be a digraph. We say that uv is a *forward arc* (or an arc) and vu is a *backward* when u dominates v. We say that W is a walk in D if its underlying graph is a walk in $G(D)$. We say that two walks are *congruent* when they have the same sequence of forward and backward arcs (so they must have the same length). An *invertible pair* is H is a pair of vertices $u, v \in V(H)$ such that the following conditions are met.

- there exists walks P and P' from u to v
- there exists walks Q and Q' from v to u
- P and Q are congruent, and P avoids Q
- P' and Q' are congruent, and P' avoids Q'

A *permutable triple* is a triple of vertices $u, v, w \in V(H)$, together with three pairs $s(x), b(x)$ for each $x \in \{u, v, w\}$ that meet the following condition for every permutation x, y, z of u, v, w.

- there exists a walk P_x from x to $s(x)$
- there exists a walk P_y from y to $b(x)$ congruent to P_x
- there exists a walk P_z from z to $b(x)$ congruent to P_y
- P_x avoids P_y
- P_x avoids P_z

A *directed asteroidal triple* (DAT for short) is a permutable triple u, v, w such that each of the three pairs $(s(x), b(x))$ is an invertible pair ($x \in \{u, v, w\}$).

2.5 Minimum Colour Sum and Optimum Cost Chromatic Partitions

In the classical colouring problem the quality of a colouring $c : V(G) \to \mathbb{N}$ is measured by the maximum colour used in (the range of) c. But in real world applications, we may encounter problems that benefit from colourings that may use more colours than the minimum number of colours possible. For example, in the scheduling problem discussed in Section 2.2, we might be interested in minimizing the total (average) wait time of the tasks instead of minimizing the maximum wait time. Again, an instance of the scheduling problem can be modelled as a conflict graph, with vertices representing the tasks and edges representing the conflicts. As before, there is a one to one correspondence between the proper colourings of the conflict graph and the scheduling of the tasks, and the wait time for every task is the colour assigned to its corresponding vertex. Hence, to minimize the total (average) wait time of the tasks, we shall find a colouring c that minimizes the sum of the colours assigned to $V(G)$.

Formally, the *minimum colour sum problem*, denoted *MCS*, is the problem that, for any input graph G together with an integer k, asks whether there exists a proper colouring of vertices of G such that the sum of all assigned colours is at most k. The minimum colour sum problem has applications in distributed resource allocation, VLSI routing and compiler design (See [10]) and was first introduced by Kubicka and Schwenk in [95]. There, they proved that the problem is NP-complete by a reduction from the chromatic number problem. Moreover, they generated a family of trees to demonstrate that for every positive integer k, there exist trees that require at least k colours to achieve the minimum colour sum. Finally, they provided a linear time algorithm for the minimum colour sum problem for trees. Later, Bar-Noy and Kortsarz proved that this problem admits no polynomial time approximation scheme, unless P = NP, even for the bipartite graphs. Their result indicates that this problem is essentially harder than the original colouring problem, which can be solved efficiently for bipartite graphs [11].

The minimum colour sum problem can be generalized by introducing costs of colours. Let G be a graph with n vertices and s_i be the cost of using colour i $(1 \leq i \leq n)$. The *optimum cost chromatic partition* of G is a proper colouring $c : V(G) \to \{1, 2, \cdots, n\}$ such that the total cost of colouring, $\sum_{u \in V(G)} s_{c(u)}$, is minimum. The decision problem *optimum cost chromatic partition problem*, denoted *OCCP*, asks whether an input graph

G together with costs s_i and an integer k admits a colouring that costs at most k. This problem was first defined by Supowit in the context of VLSI [120]. In [94], Kroon, Sen, Deng and Roy showed that when the input graph is an interval graph, the optimum cost chromatic partition problem is equivalent to the fixed interval scheduling problem with machine-dependent processing costs. They developed a linear time algorithm for trees and polynomial time algorithm for interval graphs when there are only two different values for the colouring costs. They also proved that the problem becomes NP-complete as soon as four distinct values for colouring costs are permitted, even for interval graphs.

Jansen [84] studied this problem for special classes of graphs. He proved that the problem can be solved in linear time for P_4-free graphs, and in time $O(n.log(n)^{k+1})$ for graphs with constant treewidth k. He also proved that this problem is NP-complete for bipartite graphs and permutation graphs. He has also investigated the approximability of this problem and proved that there is no polynomial time $O(n^{0.5-\epsilon})$-approximation algorithm even when the input is restricted to bipartite graphs or interval graphs, unless $P = NP$ and developed $O(n^{0.5})$-approximation algorithms for bipartite, interval and unimodular graphs.

2.6 Minimum Cost Graph Homomorphisms

Let H be a fixed graph. In the *minimum cost homomorphism problem* to H, the input is a graph G together with cost functions $c_i : V(G) \to \mathbb{N}$ for all vertices $i \in V(H)$, and the problem is to find a homomorphism f of G to H that minimizes the cost $\sum_{u \in V(G)} c_{f(u)}(u)$. For every vertex i in $V(H)$, the function c_i gives the cost of mapping vertices of G to the vertex i. We assume that the costs $c_i(u)$ are non-negative integers and refer to the summation $\sum_{u \in V(G)} c_{f(u)}(u)$ as the cost of homomorphism f. We also assume that both the input graph G and the target graph H are connected. Note that if G is disconnected, then the minimum cost for every connected component of G is an independent sub-problem that can be solved individually, and when H is disconnected, every connected component of G must map to a component of H that gives the minimum cost for that component. Moreover, there are a constant number of components in H (as H is fixed) and at most a linear number of components in G.

The minimum cost homomorphism problem for graphs was first introduced by Gutin, Rafiey, Yeo and Tso in 2006 [64] where it was motivated by a real world application of minimizing the cost of a repair and maintenance schedule for large machinery. Nevertheless,

the minimum cost homomorphism problem provides a natural and practical way to model many optimization problems. It generalizes many other problems such as list homomorphism problems, and as a consequence, list colouring problems and precolouring extension problems, retraction problems, and various optimum cost chromatic partition problems (see for example [49, 84, 85, 94]).

Gutin, Hell, Rafiey and Yeo studied the complexity of the minimum cost homomorphism problem for graphs, denoted *MinHOM(H)*, in [62]. They considered the corresponding decision version, stated below. Let H be a fixed graph with loops allowed. For an input graph G, together with homomorphism cost function c, and an integer k, decide if G admits a homomorphism to H of cost less than or equal to k. They showed that while certain minimum cost homomorphism problems have polynomial time algorithms, most are NP-complete. In particular, they showed that when H is a reflexive graph, there is a polynomial time algorithm when H is a proper interval graph; and when H is an irreflexive graph, the problem is in P when H is a proper interval bigraph. They also proved that MinHOM(H) is NP-complete in all other cases.

Theorem 2.7. *[62] Let H be a fixed graph. The minimum cost homomorphism problem to H is in* P *if every connected component of H is either a reflexive proper interval graph, or an irreflexive proper interval bigraph. In all other cases, the problem is* NP*-complete.*

It is worth noting that both the list homomorphism problem and the minimum cost homomorphism problem have the monotonicity property, in the sense that if one of the problems is NP-hard for some H, it is also NP-hard for all graphs H' that contain H as an induced subgraph. This is because one can solve the problem for H using the problem for H' by using empty lists, or relatively large costs on vertices not in H.

When H is an irreflexive graph, the proof of hardness for the minimum cost homomorphism problem (as in [64]) is by first proving that the problem is NP-complete when H is any of the forbidden structures obtained from Lemma 2.1 and then taking advantage of the monotonicity property explained above.

Lemma 2.1. *[71] A bipartite graph H is a proper interval bigraph if and only if it does not contain a cycle of length at least six, or a bipartite claw, or a bipartite net, or a bipartite tent.*

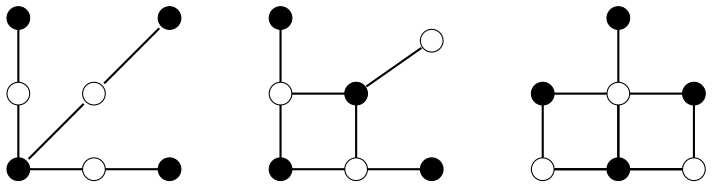

Figure 2.1: The bi-claw, bi-net and bi-tent

The bipartite claw, bipartite net and bipartite tent are depicted in Figure 2.1. When white and black vertices are distinguished, we refer to these graphs as bi-claw, bi-net and bi-tent, respectively.

The minimum cost homomorphism problems generalize the list homomorphism problems; so the hardness for large cycles follows from the hardness results on cycles in [43]. Hence, it suffices to prove that the problem is hard for the bi-claw, bi-net, and bi-tent. The authors proved that these problems are NP-complete by reductions from the problem of finding a maximum independent set in a 3-coloured graph G, which they showed to be NP-complete as an immediate result of Alekseev and Lozin [4].

In the case of reflexive graphs, the authors in [64] transformed the problem into the irreflexive case. Nevertheless, the forbidden structures that make the problem intractable for reflexive graphs are large induced cycles (with size at least four), the claw, the net, and the tent. Thus, Lemma 1.1 implies that, for a reflexive graph H, MinHom(H) is polynomial time solvable if H is a proper interval graph, and is NP-complete otherwise.

In the case of general graphs where there are both vertices with loops and vertices without loops (otherwise the graph is either irreflexive or reflexive, respectively), the problem is NP-complete by a reduction from the maximum independent set problem.

A graph (digraph) H is said to have the *min-max ordering* if its vertices can be ordered such that $u < v$, $w < t$ and $ut, vw \in E(H)$ imply that $uw, vt \in E(H)$. That is, whenever two edges cross, both the minimum ends and the maximum ends must also be edges of H. Observe that none of the forbidden structures from Lemma 2.1 admit a min-max ordering.

Spinrad and Stewart first introduced min-max ordering for bipartite graphs (which they called strong orderings in their paper) and proved that bipartite permutation graphs are exactly the bipartite graphs that admit such an ordering [119]. A *permutation graph* is a graph that has a permutation representation; each vertex is represented by an element of the permutation and two vertices are adjacent if and only if their relative order in the permutation is reversed (compared to the original order). A *bipartite* permutation graph is

a permutation graph that is also bipartite.

Later, Deng, Hell and Huang proved that the class of reflexive graphs with the min-max property coincides with the class of proper interval graphs (see [34]). Hell and Huang further improved this by showing that the class of proper interval bigraphs is the same as the class of bipartite permutation graphs, and hence the class of irreflexive graphs with the min-max property coincides with the class of proper interval bigraphs (see [70, 71]).

The polynomial algorithm for the minimum cost homomorphism problem takes advantage of the min-max property and is by a reduction to the weighted minimum cut problem. We describe the algorithm for reflexive graphs from [62]. The algorithm for bipartite graphs that admit min-max orderings is similar.

Let H be a proper interval graph, with $a_1, a_2, \cdots, a_p$ being a min-max ordering of its vertices. Denote by $\ell(i)$ the smallest subscript j such that $a_i a_j$ is an edge in $E(H)$.

Let G be the input graph of which we seek a homomorphism to H with a homomorphism cost function c. We construct an auxiliary (directed) network with vertex set $\{s,t\} \cup \{(u,i) | u \in V(G) \text{ and } a_i \in V(H)\}$ (s and t are the source and sink, respectively), and the following weighted arc set.

- an arc from the source vertex s to $(u,1)$, of weight ∞, for every $u \in V(G)$,
- an arc from (u,i) to $(u,i+1)$, of weight $c_i(u)$, for every vertex $u \in V(G)$ and every $i < p$,
- an arc from (u,p) to the sink vertex t, of weight $c_p(u)$, for every $u \in V(G)$,
- an arc from (u,i) to $(v,\ell(i))$, of weight ∞, for every edge $uv \in E(G)$ and every $i \leq p$.

The authors in [62] proved that the finite weighted cuts in the auxiliary network above correspond to the homomorphisms of G to H, and hence, the weight of the minimum cut is equal to the minimum cost homomorphism. Thus, standard (polynomial time) network flow algorithms may be employed to obtain minimum cost homomorphisms when H is a reflexive proper interval graph.

The minimum cost homomorphism problem has also been investigated for digraphs. Gurin, Rafiey and Yeo provided a dichotomy classification when H is a fixed semicomplete digraph; the problem is tractable when H is acyclic, or H is a cycle of length 2, or a H is a cycle of length 3. In all other cases, MinHOM(H) is NP-complete [63]. For general

digraphs, the same authors conjectured that MinHOM(H) is polynomial time solvable if H has an extended min-max ordering (defined below); and is NP-complete otherwise.

Let k be a positive integer and H a digraph that is homomorphic to the directed cycle $\vec{C}_k$ with vertex set $V(H) = V_1 \cup V_2 \cup \cdots \cup V_k$ where every set V_i maps to the same vertex a_i. We say that H admits a *k-min-max ordering* if there exists an ordering of every set V_i which is a min-max ordering for every induced subgraph $H[V_i \cup V_{i+1}]$ (indices modulo k). When $k = 1$ this corresponds to the usual min-max ordering of H because every digraph is homomorphism to the directed cycle $\vec{C}_1$. Digraph H admits an *extended min-max ordering* if it admits a k-min-max ordering for a positive integer k.

In [105], the authors studied the approximability of the minimum cost graph homomorphism problem when H is a bipartite graph. They showed that when the list homomorphism problem is NP-complete for some fixed bipartite graph H, then the corresponding minimum cost homomorphism problem is not approximable (with any multiplicative guarantee) unless P = NP.

Theorem 2.8. *[105] Let H be a fixed graph. If the list homomorphism problem to H is* NP*-complete, then there is no polynomial time approximation algorithm for MinHOM(H), unless* P = NP*.*

As an immediate result, they also concluded that when H contains a cycle of length at least six as an induced subgraph, no polynomial time approximation algorithm is possible.

Further, they suggested an integer linear formulation for this problem. This formulation is equivalent to the weighted network flow construction used for the polynomial algorithm in [62] discussed above. Using this equivalent formulation, they presented a randomized polynomial time 2-approximation algorithm for a subclass of proper interval bigraphs that include bipartite net and bipartite tent.

Theorem 2.9. *[105] Let H be a bipartite graph. If H admits a min ordering such that the neighbourhood of each vertex is an interval (a set of consecutive vertices in the other part of the graph), then there is a (randomized)* 2*-approximation algorithm for MinHOM(H).*

We will define min orderings in Section 4.1.

2.7 Matrix Partition of Graphs

Homomorphisms to H can be viewed as partitions of the input graph G into parts corresponding to the vertices of H. Specifically, if $x \in V(H)$ has no loop, the corresponding part P_x is an independent set in G, and if $xy \notin E(H)$, then there are no edges between the parts P_x and P_y. A further generalization of homomorphisms allows us to specify that certain parts P_x must be cliques, and between certain parts P_x and P_y there must be all possible edges.

Let M be a symmetric m by m matrix over $\{0, 1, *\}$. An *M-partition* of a graph G is a partition of vertices of G into m parts $P_1, P_2, \ldots, P_m$, such that for any two distinct vertices $u, v \in V(G)$ in (not necessarily different) parts P_i, P_j, they are adjacent when $M_{i,j} = 1$ and are not adjacent when $M_{i,j} = 0$; $M_{i,j} = *$ does not enforce any restriction. It follows that the diagonal values define the constraints on the parts; in any M-partition of G, P_i induces an independent set in G if $M_{i,i} = 0$ and a clique if $M_{i,i} = 1$. Similarly, the off-diagonal values define the constraints between two different parts; $M_{i,j} = 1$ means that every vertex in P_i is adjacent to every vertex in P_j in any M-partition. Also, $M_{i,j} = 0$ means that there are no edges in $E(G)$ joining a vertex in P_i to a vertex in P_j. We usually refer to P_i as *the i-th part*. Note that when M has no 1's, the matrix M corresponds to an adjacency matrix of a graph H, if we interpret $*$ as adjacent and 0 as non-adjacent; and in this case an M-partition of G is precisely a homomorphism of G to H. Thus M-partitions generalize homomorphisms and hence also graph colourings. Let M be a symmetric $\{0, 1, *\}$ matrix. The *graph partition problem* for matrix M, also called the *M-partition problem*, asks whether an input graph G admits an M-partition.

We say that M is a *partition matrix* when it meets the conditions mentioned above, that is, when M is a symmetric m by m matrix over $\{0, 1, *\}$. Furthermore we always assume that the rows and columns are both indexed by $\{1, 2, \cdots, m\} = [m]$. For any two sets $X, Y \subseteq [m]$ of indices of M, we denote by $M_{X,Y}$ the submatrix obtained from M by eliminating from M the set of rows that do not appear in X and the set of columns that do not appear in Y. When $X = Y$, we say that $M_{X,Y}$ is a *principal submatrix* of M (and denote it by M_X for simplicity). We denote by $\overline{M}$ the matrix obtained from M be replacing each 0 by 1 and vice versa, and call it the *complement* of M.

Matrix partition problems are frequently encountered in the study of perfect graphs. A simple example is the matrix $M = \left(\begin{smallmatrix} 0 & * \\ * & 1 \end{smallmatrix}\right)$: in this case a graph G is M-partitionable if

and only if it is a split graph. Other examples of matrices M such that M-partitions are of interest in the study of perfect graphs can be found in [50]. They include the existence of a homogeneous set [60] (M has size three), the existence of a clique cutset (M has size three), the existence of a skew cutset (M has size four), and many other popular problems [27, 50, 32, 72, 121]. If M has a diagonal $*$, it is usual for the existence problems to focus on M-partitions with all parts non-empty, otherwise the problems become trivial; this is in particular the case in the previous three examples. In any event, we emphasize the fact that M-partition problems tend to be difficult and interesting even for small matrices M. The matrices that translate some of these problems into M-partition problems are depicted in Figure 2.2.

$$\begin{pmatrix} * & 0 & 1 \\ 0 & * & * \\ 1 & * & * \end{pmatrix} \quad \begin{pmatrix} * & 0 & * & * \\ 0 & * & * & * \\ * & * & * & 1 \\ * & * & 1 & * \end{pmatrix} \quad \begin{pmatrix} * & 0 & * \\ 0 & * & * \\ * & * & 1 \end{pmatrix} \quad \begin{pmatrix} * & 0 & * \\ 0 & * & * \\ * & * & 0 \end{pmatrix}$$

Figure 2.2: Matrices for the homogeneous set, skew partition, clique cutset and stable cutset

In [50], the authors studied the graph partition problem and its list variant. They developed tools and techniques to attack these problems and used those tools to classify the M-partition and list M-partition problems for small matrices M. In particular, we refer to *sparse-dense* partitions, which we will cover in Chapter 3. They presented a dichotomy for the M-partition problem when M has size four (Theorem 2.11) and for the list version when M has size three (Theorem 2.10). Moreover, they found classifications for the list version when M has size 4 but does not have any $*$ on the main diagonal. Also, for arbitrary large matrices, their results include a complete classification for $*$-free matrices and a reduction to $n^{O(log(n))}$ H-colouring problems (Theorems 2.12, 2.13). In the rest of this Section, we assume that M is a fixed matrix of size m over $\{0, 1, *\}$.

Theorem 2.10. *[50] Suppose $m = 3$. Then the list M-partition problem is* NP*-complete when M or its complement is the matrix for* 3*-colouring or the stable cutset, and is polynomial time solvable otherwise.*

Theorem 2.11. *[50] Suppose $m = 4$. The M-partition problem (without lists) is* NP*-complete when M has no diagonal $*$s and contains the matrix of* 3*-colouring or its complement, and is polynomial time solvable otherwise.*

Theorem 2.12. *[50] If M is a $(0,1)$-matrix, then the list M-partition problem is polynomial time solvable.*

Theorem 2.13. *[50] Each list M-partition problem can be reduced to $n^{O(log(n))}$ instances of the list H-colouring problems (each with possible additional restrictions on the allowed lists).*

Later, Cameron et al. in [24] improved their results, and classified the list M-partition problem for all remaining matrices of size 4, except for the matrix for the stubborn problem, and its complement. Recently, Cygan et al. solved the last remaining case, the stubborn problem, by providing a polynomial time algorithm, and hence completed the classification of the list M-partition problem for matrices of size four, as stated in Theorem 2.14 below.

Theorem 2.14. *[30] Suppose $k = 4$. The list M-partition problem is solvable in polynomial time, except when M contains the matrix for 3-colourability, stable cutset, or their complements, or M is the matrix for stable cutset pair, $2K_2$, or their complements, in which case the problem is* NP*-complete.*

In addition to the general graph partition problems mentioned above, in which the input graph is not restricted, there are results on some restricted versions. Among which, we may refer to the the graph partition problem considered for chordal graphs, perfect graphs or co-graphs (see [45, 51, 44] for some examples). The directed versions have also been studied (see [52, 80, 79]).

2.8 Counting Graph Homomorphisms

Another related problem that has been studied widely is the counting problem for graph homomorphisms (and graph colourings). Let H be a fixed graph. The problem of *counting homomorphisms* to H, also called counting H-colourings and denoted $\#\,\mathrm{HOM(H)}$, asks for the number of homomorphisms of an input graph G to a fixed graph H. Homomorphism numbers can be used to describe large parts of extremal graph theory, and they can be used to characterize graphs (See [17]). This problem has also applications in statistical physics; cf. [38].

It is well known that counting the number of graph k-colourings is polynomial when $k < 3$ and is $\#\,$P-complete otherwise. Dyer and Greenhill proved that counting the number

of graph homomorphisms is #P-complete in general, with polynomial algorithms possible only in trivial cases when every connected component of H is a complete reflexive graph, or a complete irreflexive bipartite graph [38] (Theorem 2.15). They also proved that when counting homomorphisms is NP-complete for some graph H, then there exists a constant Δ for which the problem remains NP-complete for input graphs with maximum degree Δ, and showed that in some practical cases $\Delta = 3$.

Theorem 2.15. *[38] Let H be a fixed graph. The problem of counting H-colourings of graphs is #P-complete if H has a connected component which is not a complete reflexive graph or a complete bipartite graph. Otherwise, the problem is polynomial time solvable.*

Later, Hell and Nešetřil proved a similar result for counting list homomorphisms of graphs.

Theorem 2.16. *[76] Let H be a fixed graph. #ListHOM(H) is #P-complete if H has a connected component which is not a complete reflexive graph or a complete bipartite graph. Otherwise, the problem is polynomial time solvable.*

Chapter 3

Counting Complexity

3.1 Introduction

It is well known that the number of bipartitions of a graph G can be computed in polynomial time. Indeed, we can first check, in polynomial time, if G is bipartite, and if not, the answer is 0. If G is bipartite, we can find, in polynomial time, the number c of connected components of G. Since each such component admits exactly two bipartitions, the answer in this case is 2^c. Interestingly, the number of bipartitions can also be computed using linear algebra: if each vertex v is associated with a variable x_v over the field $F_2 = \{0, 1\}$, and each edge uv with the equation $x_u + x_v = 1$ in F_2 (i.e., modulo 2), then the number of solutions of this system is precisely the number of bipartitions of G. Thus 2-colourings of graphs can be counted in polynomial time. It is also known that the counting problem for m-colourings of G with $m > 2$ is $\#$ P-complete [99]. This is the *dichotomy* of the counting problems for graph colourings.

Dichotomy of counting homomorphisms to graphs H has been established by Dyer and Greenhill [38]. Namely, if each connected component of H is either a reflexive complete graph, or an irreflexive complete bipartite graph, then counting homomorphisms to H can be solved by trivial methods, as in the above example (or, once again, by linear algebra). In all other cases, counting homomorphisms to H is $\#$ P-complete [38]. Other dichotomies for homomorphism counting problems, in bounded degree graphs, or for homomorphisms with lists, are discussed in [76]. (In the list version of the problem, the graph G has a list $L(v) \subseteq V(H)$ for each vertex $v \in V(G)$ and only homomorphisms f that satisfy $f(v) \in L(v)$, for all $v \in V(G)$, are counted.) In particular, it is proved in [76], that, as without lists, if each

connected component of H is either a reflexive complete graph, or an irreflexive complete bipartite graph, then counting list homomorphisms to H can be solved by easy polynomial time methods, and in all other cases counting list homomorphisms to H is $\#$ P-complete [76].

We note for future reference the following *complementarity* of M-partitions. Denote by $\overline{M}$ the matrix obtained from M be replacing each 0 by 1 and vice versa. Then an $\overline{M}$-partition of a graph G is precisely an M-partition of the complement $\overline{G}$.

In the literature there are several papers dealing with algorithms and characterizations of graphs admitting M-partitions (or list M-partitions) [30, 31, 50, 32]; these are detailed in a recent survey [67], cf. also a slightly older survey [78], or the book [77]. We focus on the counting problem for M-partitions. Recall that for homomorphism problems, except for trivial cases, counting homomorphisms turned out to be $\#$ P-complete [38]. More generally, the complexity of counting solutions of constraint satisfaction problems has also been classified [22]. We contrast these facts by exhibiting several counting problems for M-partitions, where highly combinatorial methods seem to be needed. In the process, we completely classify the complexity of counting the number of M-partitions for all matrices of size less than four.

Given a matrix M, we want to know how hard it is to count the number of M-partitions. As a warm-up, we prove the following classification for two by two matrices $M = \begin{pmatrix} a & c \\ c & b \end{pmatrix}$.

Theorem 3.1. *If c and exactly one of a, b is $*$, then the problem of determining the number of M-partitions is $\#$ P-complete. Otherwise there is a polynomial algorithm to count M-partitions.*

Proof. If, say, $a = c = *$ and $b = 0$, then the number of M-partitions of G is precisely the number of independent sets in G, which is known to be $\#$ P-complete [114]. Similarly, if $a = c = *$ and $b = 1$, we are counting the number of cliques in G, which is also $\#$ P-complete [114] (or by complementarity).

If M has no 1, the result follows by [38], so we assume that M contains at least one 1, and, by complementarity, also at least one 0. If $c = 0$, the two parts have no edges joining them, and at least one part is a clique. The number of M-partitions can easily be determined once the connected components of G have been computed. (For instance if $a = 1$, $b = *$, and t connected components of G are cliques, then G has t M-partitions. When $a = 1$, $b = 0$, the counting is even easier.) If $c = 1$, the result follows by complementation.

Thus we assume that $c = *$. By symmetry, we may assume without loss of generality that $a = 0$ and $b = 1$. Such an M-partition of G is called a *split partition*. It follows from [50] that the number of split partitions is polynomial, and can be found in polynomial time (see Theorem 3.1 in [50]). Therefore they can also be counted in polynomial time.

□

The two matrices $\left(\begin{smallmatrix} * & * \\ * & 0 \end{smallmatrix}\right)$ and $\left(\begin{smallmatrix} * & * \\ * & 1 \end{smallmatrix}\right)$ from the first paragraph of the proof will play a role in the sequel, and we will refer to the as the *matrices for independent sets, and cliques*.

3.2 Decomposition Techniques

The last two by two matrix, corresponding to split partitions, illustrates the fact that there are interesting combinatorial algorithms for counting M-partitions. In this section we examine a few other examples, in this case of three by three matrices, documenting this fact.

In the remainder of this part, we assume we have the matrix $M = \begin{pmatrix} a & d & e \\ d & b & f \\ e & f & c \end{pmatrix}$.

We begin by discussing some cases related to the example of split partitions at the end of the previous section. In fact, the general technique of Theorem 3.1 from [50] is formulated in the language of so-called *sparse-dense partitions* as follows.

Let G be a graph with n vertices, and $\mathcal{S}$ and $\mathcal{D}$ be two families of subsets of $V(G)$ that meet the following conditions.

- They are closed under taking subsets,
- There exists a constant c such that for every $S \in \mathcal{S}$ and $D \in \mathcal{D}$, the intersection $S \cap D$ has size at most c,
- They can be recognized in polynomial time.

Then there are at most n^{2c} partitions of $V(G)$ into two sets S and D with $S \in \mathcal{S}$, $D \in \mathcal{D}$, and all those partitions can be generated in polynomial time ($\mathcal{O}(n^{2c+2})$) [50]. For split partitions, we take $\mathcal{S}$ to be all independent sets, $\mathcal{D}$ all cliques, and $c = 1$. If we take for $\mathcal{S}$ all bipartite induced subgraphs, for $\mathcal{D}$ all cliques, and $c = 2$, we can conclude that any graph G has only a polynomial number of partitions $V(G) = B \cup C$, where B induces a bipartite

graph and C induces a clique, and all these partitions can be generated in polynomial time. This result is sufficient to cover a number of polynomial cases.

Theorem 3.2. *If a, b, c are not all the same and none is $*$, then the number of M-partitions can be counted in polynomial time.*

Proof. Up to symmetry and complementarity we may assume that $a = b = 0, c = 1$. For each sparse-dense partition $V(G) = S \cup D$, we shall test how many partitions of the subgraph induced by S, into the first part and the second part, satisfy the constraints induced by the entries d, e, and f of the matrix M. We impose the constraints due to e and f by introducing lists on the vertices in S. (If, say, $e = 1$, then only vertices completely adjacent to D will have the third part in their lists, and similarly for other values of e and f.) If d is 0 or $*$, this corresponds to counting list homomorphisms to an empty graph (I_2) or a complete bipartite graph (K_2), both of which are polynomial by [50], as noted above. When $d = 1$, there are at most two different partitions of S to consider, so we can check these as needed. □

The next result deals with a class of problems related to homogeneous sets and modular decomposition [60]. A *module* in a graph G is a subset A of its vertices with the following property. Every vertex not in A is either adjacent to every vertex in A, or is not adjacent to any of them. Every vertex of G is clearly a module, as is the empty set, and the whole set $V(G)$. These are called *trivial* modules. A graph is *prime* when all of its modules are trivial. *Non-trivial* modules are also often called *homogeneous sets.*

Two modules *overlap* if their intersection is not empty and none of them is a subset of the other one. A non-empty module is a *strong* module if it does not overlap with any other module, and is a *weak* module otherwise. Let $S = \{S_1, S_2, \cdots, S_k\}$ be the family of strong modules of G. The definition of the strong modules implies that for any two distinct strong modules S_i and S_j, one of the following three cases must occur.

- S_i and S_j are disjoint (That is, $S_i \cap S_j = \emptyset$)
- $S_i \subset S_j$
- $S_j \subset S_i$

This *containment* relation can be used to define a tree on S with its root being $V(G)$: Consider a vertex for every member of S, and join two vertices if and only if one is the

smallest superset of the other in S. This is called the *modular decomposition tree* of G and is denoted by $MD(G)$. It turns out that the leaves of the tree are precisely the singleton subsets of S (which correspond to vertices of G).

If G is disconnected, then every component of G is a module of G as it is not adjacent to any other vertex in G. Interestingly, it turns out that every such module is a maximal strong module, and hence, its corresponding vertex in the modular decomposition tree is adjacent to the root vertex corresponding $V(G)$. Similarly, when $\overline{G}$ is disconnected, maximal strong modules of G are precisely the co-components of G. It turns out that when both G and $\overline{G}$ are connected, there is also a decomposition of $V(G)$ into maximal strong modules. In fact, Gallai proved that there is always a unique representation of G by its maximal strong modules (up to order) as stated in the following theorem.

Theorem 3.3 ([55])**.** *For any graph G one of the following three cases must occur.*

1. *G is disconnected, with components $G_1, G_2, \ldots G_k$.*

   Each union of the sets $V(G_i)$ is a module of G, and the other modules of G are precisely all the modules of individual components G_i.

2. *The complement of G is disconnected, with components $H_1, H_2, \ldots, H_\ell$.*

   Each union of the sets $V(H_j)$ is a module of G, and the other modules of G are precisely all the modules of individual subgraphs $\overline{H_j}$.

3. *Both G and its complement are connected.*

   There is a partition $S_1, S_2, \ldots, S_r$ of $V(G)$ such that all the modules of G are precisely all the modules of individual subgraphs induced by the sets S_t, $t = 1, \ldots, r$, plus the module $V(G)$.

We will call the sets $V(G_i)$, $V(\overline{H_j})$, and S_t from the theorem the *blocks* of G. According to the theorem, all modules are modules of the blocks, except for modules that are unions of (at least two) blocks when G or its complement is disconnected. We call these latter modules *cross modules*. Also, we distinguish two types of nodes in $MD(G)$; *prime* nodes and *degenerate* nodes. A node is prime when no union of its children is a module, and is degenerate otherwise (which by the above theorem, means that every union of its children is a module).

Based on this theorem, one can recursively decompose any graph into modules. Even though the number of modules of G can be exponential (for instance if $G = K_n$), the modular decomposition tree has polynomial size and can be computed in linear time [65]. The first polynomial-time algorithm ($\mathcal{O}(n^4)$) for computing the modular decomposition tree was developed by James et al. [83] in 1972. Since then, a number of algorithms were published that improved the running time, including linear-time algorithms [29, 108, 109, 124]. For a recent survey on algorithmic aspects of modular decomposition see [65].

For our purposes in this thesis, we take advantage of existing linear-time algorithms for computing the modular decomposition tree, by making calls to a subroutine called `MDTree`. We assume that `MDTree` takes as input a simple graph G, and returns $MD(G)$, the modular decomposition tree of G.

We note that there is a natural linear time algorithm that counts the non-empty modules directly on the modular decomposition tree. Let $T(G)$ denote the total number of non-empty modules of a graph G. We first compute the decompositions (1, 2, or 3) in Theorem 3.3. In cases 1 and 2, we have $T(G) = 2^t - t - 1 + \sum T(B)$, where t is the number of blocks, and the sum is over all blocks B. In case 3, we have $T(G) = 1 + \sum T(B)$. Indeed the number of cross modules is 1 in the case 3 (only the module $V(G)$ is a cross module), and is $2^t - t - 1$ in the other two cases (subtracting one for the empty set and one for each individual block). Since the sizes of the blocks for the recursive calls sum up to n, this yields a recurrence for the running time whose solution is linear in n. This shows that the number of modules of G can be counted in polynomial time. This algorithm is depicted in Algorithm 3.1.

Corollary 3.1. *Let G be a simple graph. Algorithm 3.1 runs in linear time and computes the number of non-empty modules of G.*

Once the non-empty modules are counted, counting M-partitions becomes easy. Indeed, counting the number of M-partitions when the first part is empty is trivial; there are exactly 2^n partitions of G ($n = |V|$) into the second and third parts, as there is no restriction on the parts, or between them.

Corollary 3.2. *If d, e, f are all different, and $a = b = c = *$, then the number of M-partitions can be computed in polynomial time.*

We note that a number of variants can be counted the same way. Consider, for instance, the number of modules that are independent sets. In case 1, if s of the blocks consist of a

Input : Simple graph G and its modular decomposition tree, $md =$ `MDTree(`G`)`
Output: Number of modules of G

```
1  count ← 0
2  t ← md.children
3  if |V(G)| = 0 then
4  |  return 0
5  end
6  if t = 0 then
7  |  return 1
8  end
9  if md.type = prime then
   |  // there are no cross modules in a prime node
   |  // count the module that corresponds to the union of all blocks
10 |  count ← 1
11 else
   |  // counting cross modules: unions of (at least two) blocks
12 |  count ← (2^t − t − 1)
13 end
   // counting modules of the blocks recursively
14 for s ← 1 to t do
15 |  count ← count + CountModules(md.child[i])
16 end
17 return count
```

Algorithm 3.1: `CountModules`

single vertex, then the number of cross modules changes to $2^s - s - 1$. In case 2, as well as 3, there are no cross module in this case (unless G has no edges). Moreover, it is easy to see that the number of non-empty independent modules of G inside a block B is precisely the number of non-empty independent modules of the graph induced by B, in all three cases. Thus the number of independent non-empty modules of G is $T(G) = 2^s - s - 1 + \sum T(B)$ in case 1, and $T(G) = \sum T(B)$ in cases 2 and 3. Of course, the number of modules that are cliques can be counted in a similar way, or by looking at the complement.

Consider next counting the number $T(G)$ of non-empty modules S such that the set R of vertices adjacent to S is a clique, and the set Q of vertices non-adjacent to S is independent. (We say briefly that S satisfies the restriction that it *has neighbours in a clique and non-neighbours in an independent set.*) To count the number of non-empty cross modules satisfying the restriction in case 1, we assume again that there are t blocks G_i and that s of them consist of a single vertex, and for simplicity assume that $s < t - 1$. (If $s = t, t - 1$, an obvious modification will work.) Then the number of non-empty cross modules that satisfy the restriction (as there are no neighbours, the restriction in this case only requires the non-neighbours to form an independent set) is 2^s. In case 2, again assuming there are $s < t - 1$ blocks $V(\overline{H_j})$ that are singletons, we obtain the same number of non-empty cross modules that satisfy the restriction, 2^s. (And once again, in case 3, there is just one cross module.) Consider now the number of modules of a block B that satisfy the restriction. First of all, the block B itself must satisfy the restriction (of having neighbours in a clique and non-neighbours in an independent set), otherwise no module of B will satisfy the restriction in G. Because of this, it is easy to see that any module of B satisfying the restriction in B will also satisfy the restriction in G. We obtain the recurrences $T(G) = 2^s + \sum T(B)$ in case 1 and 2, and $T(G) = 1 + \sum T(B)$ in case 3. In both cases, the summation is taken over all blocks B that satisfy the restriction.

As a last example, we consider counting the number $T(G)$ of non-empty modules S such that S, as well as its set R of neighbours, and its set Q of non-neighbours in G, are independent sets. The number of cross blocks is zero in case 1, unless G has no edges. In case 2, it is also zero, unless G is a complete bipartite graph, in which case it is 2. In case 3, it is again zero, unless G has no edges. As before, it is easy to see that only modules of a block B that itself satisfies the restriction are to be counted. Suppose B satisfies the restriction and has a non-empty set of neighbours in G. Then only modules of B that have an empty set of neighbours in B count as modules of G. Indeed, a module with a neighbour

in B together with a neighbour of B in G would not form an independent set. Such modules of B only exist (and are easy to count) if B has no edges. If B itself has no neighbours, then B is an independent set and any subset of it is a module. With these observations, the number of non-empty modules of this type are also easy to count in polynomial time.

We can in fact handle all restrictions of this type on the module S, its set of neighbours R, and its set of non-neighbours Q. The above examples are respectively: (i) S, R, Q unrestricted; (ii) S independent, R, Q unrestricted; (iii) S a clique, R, Q unrestricted; (iv) S, Q, R independent; and (by complementation) (v) S, Q, R cliques. It turns out that all the remaining combinations of restrictions (independent, clique, or unrestricted) for the sets S, Q, R can be treated in similar ways. Our examples illustrate all the kinds of arguments needed in these proofs.

Note that the arguments are written so as to count the number of non-empty modules of the various kinds. Of course by adding 1, we can count all such modules.

Theorem 3.4. *There is a polynomial time algorithm to count the number of modules satisfying any combination of restrictions where the module itself, its set of neighbours, and its set of non-neighbours are an independent set, a clique, or unrestricted.* □

It is easy to see that each restricted kind of module corresponds to an M-partition in which $d = 1, e = 0, f = *$, and a is determined by the constraint on S ($a = 0$ if R is to be independent, $a = 1$ if it is to be a clique, and $a = *$ if S is unrestricted), b is determined by the constraint on R, and c by the constraint on Q.

Corollary 3.3. *If $d = 1, e = 0, f = *$, then the number of M-partitions with non-empty first part can be computed in polynomial time.*

However, the number of M-partitions must also take into account the partitions with the first part empty. By applying Theorem 3.1, we conclude that the number of M-partitions with empty first part can also be counted in polynomial time, unless the second and third part form the matrix for independent sets or cliques.

Theorem 3.5. *If d, e, f are all different, and M does not contain, as a principal submatrix, the matrix for independent sets or cliques, then the number of M-partitions can be counted in polynomial time.* □

The last kind of decomposition refers to the matrices M with two off-diagonal 0s. In

these cases, M can be viewed as consisting of two submatrices and the M-partition problem can be reduced to the corresponding problems for these two matrices.

Theorem 3.6. *Assume that two of d, e, f are 0 and M does not contain as a principal submatrix the matrix for independent sets or cliques. Then counting the number of M-partitions is polynomial.*

Proof. We may assume that there is at least one 1 in M. Otherwise, the result follows from [38], since the classification is given in terms of connected components. Recall that for any subset $S \subset \{1, 2, 3\}$, M_S denotes the principal submatrix of M on rows and columns specified by S. Without loss of generality, let $d = e = 0$ and assume $M_{\{2,3\}}$ is not the matrix for independent sets or cliques.

If $a = 1$, then only a clique component can be placed in P_1. The clique components of G can be found in polynomial time, there is only a linear number of them, and for each of these candidates, the number of $M_{\{2,3\}}$-partitions can be counted in polynomial time by Theorem 3.1.

If $f = 1$, in any partition of G with both P_2 and P_3 non-empty, only one component of G can be placed into $P_2 \cup P_3$, because every vertex in P_2 must be adjacent to every vertex in P_3. Hence, counting non-trivial partitions is polynomial: for each connected component C of G, multiply the number of partitions of C to $M_{\{1\}}$ by the number of partitions of $G - C$ to $M_{\{2\}}$ (which is one, unless $a = 1$ and C is not a clique, or $a = 0$ and C is not an independent set, in which case it is zero). Counting partitions with empty sets P_2 or P_3 is also polynomial, by Theorem 3.1.

If $f = 0$, then there are no edges between the parts, and hence, every connected component of G must be placed entirely into one part in any M-partition. Moreover, M has at least one 1 that means at least one part is a clique. Thus, the M-partitions of G can be counted in polynomial time once the connected components of G are known (similar to the case where $a = 1$).

For the remaining cases, we can assume that $a \neq 1$ and $f = *$. We can also assume without loss of generality that $b = 1$ (up to symmetry) and $c \neq *$ (as $M_{\{2,3\}}$ is not the matrix for cliques). If $c = 1$, at most two connected components of G can be placed into $P_2 \cup P_3$, thus, all of the candidates can be found in polynomial time. For each of these candidates (which are the empty set, any single component, or any two components of G), counting the number of $M_{\{2,3\}}$-partitions is easy by Theorem 3.1, and deciding whether the

rest of the graph can be placed into P_1 is trivial.

In all of the remaining cases, $c = 0$. This case is similar to the previous case ($c = 1$) when G does not have any isolated vertices, as the maximum number of components that can be placed into $P_2 \cup P_3$ is one. We handle the case that G has isolated vertices separately. Assume that G has t isolated vertices and denote by G' the maximal subgraph of G with no isolated vertices. If G' is empty (with no vertices), at most one vertex can be in P_2, and the number of M-partitions is simply $2^t + t * 2^{t-1}$. If G' is not empty, it has at least one edge, and so, at least one vertex in P_2. This further implies that no isolated vertex of $G - G'$ can be in P_2 and there are exactly 2^t M-partitions of G for every M-partition of G'. □

3.3 Counting Split Partitions

Our final example of a polynomial counting problem deals with the following matrix

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

First, we present a polynomial time algorithm for counting M-partitions of bipartite input graphs in Theorem 3.7. Then we show that when the input graph is not bipartite, the problem can be reduced in polynomial time to the problem with bipartite inputs in Theorem 3.8.

Theorem 3.7. *The number of M-partitions of bipartite input graphs can be computed in polynomial time.*

Let $G = (X, Y)$ be a bipartite graph. Parts X, Y are independent sets, and thus, none of them can have vertices in parts P_2 and P_3 simultaneously; each part must be placed either entirely in $P_1 \cup P_2$ or entirely in $P_1 \cup P_3$, and we focus, without loss of generality, on the case when the vertices of X are placed in $P_1 \cup P_2$, and the vertices of Y in $P_1 \cup P_3$.

A *universal vertex in a bipartite graph* is a vertex that is adjacent to all vertices in the opposite part of the graph. A graph is $2K_2$*-free* if it does not contain an induced $2K_2$. It is shown in [100] that a $2K_2$-free bipartite graph G contains a universal vertex.

Lemma 3.1. *Let uv be an edge in an induced $2K_2$ in a graph G. In any proper M-partition of G, exactly one of the vertices u or v is placed in part P_1.*

Proof. Vertices u and v must not be placed in the same part, because they are adjacent. Assume for a contradiction that they are placed to P_2 and P_3 respectively. Let $u'v'$ be the other edge in the $2K_2$. Note that u' and v' are both nonadjacent to u and v (in parts P_2 and P_3), so they can neither be in P_2 nor in P_3. So, both must be in P_1, which is impossible, as they are adjacent. □

Suppose we have a partial M-partition f of G, i.e., a partition of an induced subgraph of G. In other words, some vertices v of G have an assigned part $f(v)$; moreover, vertices $u \in X$ have $f(u) = P_1$ or $f(u) = P_2$ and vertices $v \in Y$ have $f(v) = P_1$ or $f(v) = P_3$. This partial M-partition can be expanded uniquely as long as one of the following conditions is met.

- There are vertices in G adjacent to some vertices in part P_1.
- There are vertices in G nonadjacent to a vertex of different colour (X or Y) in P_2 or P_3.

This is because, when there exist a vertex $u \in X$ that is adjacent to a vertex $v \in Y$ with $f(v) = P_1$, it can only be assigned to part P_2 because part P_1 must be an independent set. Similarly, when $v \in Y$ is adjacent to any $u \in X$ with $f(u) = P_1$, v can only be assigned to part P_3. Furthermore, parts P_2 and P_3 must be fully adjacent, hence, any unassigned vertex $u \in X$ that is not adjacent to some $v \in Y$ with $f(v) = P_3$ can only be assigned to part P_1, and any unassigned vertex $v \in Y$ that is not adjacent to some $u \in X$ with $f(u) = P_2$ can only be assigned to part P_1 as well.

It is easy to see that both types of expansions explained above can be tested in polynomial time (no more than, say, $\mathcal{O}(n^3)$). After each step, we try to find a new vertex to expand the partial M-partition and we continue this process of assignment until every remaining vertex in part X (respectively Y) is fully adjacent to all vertices in P_3 (respectively in P_2), and is not adjacent to any vertex in part P_1. (Otherwise we can continue to extend the partition). Let f be a proper partial partition, and f^* be the final extension using the above rules. It is easy to see that the above extension assigns a unique part to all assigned vertices. Hence, we have the following lemma.

Lemma 3.2. *The number of partitions of G consistent with the partial partition f, is precisely the number of partitions of $G - D$, where D is the domain of the final extension f^*.*

We now complete the proof of Theorem 3.7.

Proof. First, assume that G has a universal vertex u. If u is assigned to P_1, then all vertices from the other part of the bipartition of G must be placed entirely in P_3 when $u \in X$, or entirely in P_2 when $u \in Y$. Also, only remaining universal vertices (that have the same colour as u) may go to P_2 (respectively, to P_3). This gives a total of 2^k possible solutions, where k is the number of remaining universal vertices. If u is placed in P_2, then every solution is also a solution for $G - u$. In this case, let $G_1 = G - u$ and let G_2 be the empty graph. Notice that finding a universal vertex, if any, and constructing G_1 and G_2 can be performed in $\mathcal{O}(n + m)$.

Second, assume that G does not have a universal vertex, and hence contains an induced $2K_2$. We can find such an induced $2K_2$ in polynomial time. Let uv be an edge of a $2K_2$ and assume without loss of generality that $u \in X$. There are two ways to assign u and v to parts in M. Either u is in part P_1 and v is in P_3, or u is in P_2 and v is in P_1. We shall count the number of partitions by extending both cases. Let X_1 and Y_1 be the set of remaining vertices (from parts X and Y, respectively), after extending the first case, and X_2 and Y_2 be the set of remaining vertices after extending the second case. Let $G_1 = G[X_1 \cup Y_1]$ and $G_2 = G[X_2 \cup Y_2]$. We can count the number of partitions recursively on subgraphs G_1 and G_2. Observe that X_1 is fully adjacent to v, and X_2 is not adjacent to v. Hence X_1 and X_2 are disjoint. Similarly, Y_1 and Y_2 are also disjoint. These can also be done in polynomial time (at most $\mathcal{O}(n^4)$ in the trivial way).

In both cases, the maximum time needed to count the solutions can be computed using the following recursive formula.

$$t(n) = t(n_1) + t(n_2) + O(n^r)$$

where n_1 is the size of G_1, n_2 is the size of G_2, and n^r is an upper bound for the time needed to find G_1 and G_2 and calculate the final result (where r is a constant less than or equal to 5). Observe that in both cases, $n_1 + n_2 < n$, so $t(n) \in O(n^{r+1})$. □

We are ready to prove the main result of this part, Theorem 3.8.

Theorem 3.8. *The number of M-partitions of any graph G can be computed in polynomial time.*

Proof. By Theorem 3.7, we may assume that G is not bipartite. If it does not contain a triangle, then the shortest odd cycle has at least five vertices. In such a case, the number of M-partitions of G is zero. Indeed, the largest complete bipartite subgraph of the cycle has three (consecutive) vertices, and hence at least two adjacent vertices of the cycle must be placed in P_1 in any M-partition of G, which is impossible.

Otherwise, we find a triangle uvw in G, in polynomial time, and then add the numbers of M-partitions of G with the six possible assignments of u, v, w to P_1, P_2, P_3. (Since the parts are independent, u, v, w must be placed in distinct parts.) For each such assignment, we shall first extend the assignment by placing vertices that are forced to certain parts uniquely.

In the first phase, we proceed as follows:

- a vertex with neighbours in two different parts is placed in the third part;
- a vertex with a non-neighbour in P_2 as well as a non-neighbour in P_3 is placed in P_1;
- a vertex with both a neighbour and a non-neighbour in P_2 (respectively P_3) is placed in P_1.

It is clear that these are forced assignments in any M-partition of G extending the given assignment on u, v, w. If at any time these assignments (or those below) violate the requirements of an M-partition, the corresponding count is zero.

The first rule ensures that every remaining vertex is adjacent to at most one part, while the second rule ensures that it is adjacent to at least one vertex in $P_2 \cup P_3$. Hence, after the first phase, every vertex is either fully adjacent to P_2 and not adjacent to any vertex of $P_1 \cup P_3$, forming a set called X, or is fully adjacent to P_3 and not adjacent to any vertex of $P_1 \cup P_2$, forming a set called Y. Note that the vertices of X must be placed in $P_1 \cup P_3$ and the vertices of Y in $P_1 \cup P_2$.

In the second phase, we extend the assignment using the following rules. (In brackets we explain why these rules are forced.)

Assume uv is an edge with $u, v \in X$, and w is any vertex in Y. (Symmetric rules apply for edges uv with $u, v \in Y$, and any $w \in X$.)

- If w is adjacent to both u and v then w will be placed in P_2. (This is forced because u and v must be in different parts P_1 and P_3, and w is adjacent to both of them.)

- if w is nonadjacent to both u and v, then w is placed in P_1. (This is forced because u and v must be in different parts P_1 and P_3, and w is not adjacent to either of them, so it is nonadjacent to at least one vertex in P_3 in the final partition.)
- if w is adjacent to u but not to v, then u is placed in P_3 and v is placed in P_1. (This is forced because if $v \in P_3$ then $u \in P_1$ and $w \in P_2$, which is impossible as vw is not an edge of G.)

When there are no more choices remaining, either we have X or Y empty, or the graph induced by $X \cup Y$ is bipartite. The former case corresponds to partitions of the remaining vertices into two of the three parts, counted in polynomial time by Theorem 3.1. The latter case is counted by Theorem 3.7. □

3.4 The Dichotomy for Small Matrices

Our main result in this section is the following dichotomy. Notice that when M does not contain any 1's (or does not contain any 0's), then the matrix M represents a simple graph (or its complement) and hence, the dichotomy follows from [38].

Theorem 3.9. *Suppose M is a partition matrix of size m with $m < 4$, and assume M contains both a 0 and a 1.*

If M contains, as a principal submatrix, the matrix for independent sets, or the matrix for cliques, then the counting problem for M-partitions is $\#\,\mathrm{P}$-complete.

Otherwise, there is a polynomial time algorithm to count the number of M-partitions.

Proof. We first derive the polynomial cases from our existing results. We assume throughout that M contains both a 0 and a 1.

For $m = 2$ this follows from Theorem 3.1. Thus we assume that $m = 3$, say $M = \begin{pmatrix} a & d & e \\ d & b & f \\ e & f & c \end{pmatrix}$, and M does not contain as a principal submatrix the matrix for independent sets, or the matrix for cliques. If d, e, f are all different, then the number of M-partitions is computed by Theorem 3.5. If at least two of d, e, f are 0 (or at least two are 1), then M-partitions are counted by Theorem 3.6.

Hence, in all the remaining cases, we may assume that $d = e = *$, $f \neq 0$, by symmetry and complementarity. Notice that M contains both a 0 and a 1; hence, $f \neq 0$ implies that

at least one of a, b, c is 0. Moreover, $d = *$ implies that $a = *$ if and only if $b = *$, because M does not contain the principal submatrices for independent sets or cliques, and similarly, $e = *$ implies that $a = *$ if and only if $c = *$. Thus, we note that none of a, b, c is $*$ and we conclude by Theorem 3.8 ($f = 1$, $a = b = c = 0$) or Theorem 3.2 (otherwise) that the number of M-partitions can be computed in polynomial time.

It remains to show that if M contains, as a principal submatrix, the matrix for independent sets or cliques (cf. the definition at the end of Section 3.1), then counting M-partitions is # P-complete. We shall again use the notation $M' = \left(\begin{smallmatrix} b & f \\ f & c \end{smallmatrix}\right)$. We shall assume that M' is the matrix for independent sets, without loss of generality, say, that $b = f = *$, $c = 0$.

We consider two distinct cases, depending on the value of a. Our proof is completed by the following two lemmas. □

For the purposes of the lemmas we introduce two constructions. The *universal vertex extension* G^* of a graph G is a graph obtained from G by adding a new vertex u, adjacent to all vertices of G. The *isolated vertex extension* G^o of G is a graph obtained from G by adding a new isolated vertex u.

Lemma 3.3. *If $a \neq 0$, $b = f = *$, $c = 0$, then counting the number of M-partitions is # P-complete.*

Proof. We reduce from the number of independent sets in graph G using the isolated vertex extension of G. Let $\#I(G)$ denote the number of independent sets of G, and $\#M(G)$ the number of M-partitions of G.

Given an input graph G, we first construct G^o. We count the number of M-partitions of G^o according to the placement of the isolated vertex u. When $a = 1$, we consider the two following cases:

If both d, e are different from 1, then $\#M(G^o) = \#I(G) + 2\#M(G)$. Indeed, if u is placed in P_1, then all other vertices must be placed in P_2 or P_3 (since $a = 1$ and u is an isolated vertex). The number of such M-partitions is exactly $\#I(G)$: any independent set in G will be placed in P_3 and the remaining vertices in P_2. The number of M-partitions with u placed in P_2 or P_3 is $\#M(G)$ each, since there is no restriction on where the other vertices will be. This gives $2\#M(G)$.

Otherwise, assume $d = 1$. (The argument for $e = 1$ is analogous). The number of M-partitions of G^o with u is placed in P_1 is 0 or 1 (the latter only if G has no edges), since the vertices of G must not be placed in P_2. If u is placed in P_2, then all vertices of G must

be placed in P_2 and P_3, and there are exactly $\#I(G)$ ways to do that. Placing u in P_3 does not restrict any other vertices, and we will have $\#M(G)$ possible partitions. Hence $\#M(G^o) = \#I(G) + \#M(G) + k$ where k is 0 or 1 as explained above.

The argument is similar when $a = *$.

If $d = 1, e = *$, then $\#M(G^o) = 2 \times \#I(G) + \#M(G)$. This is because the number of M-partitions with u in P_1 or P_2 is equal to the number of independent sets of G (because P_2 or P_1 respectively must be empty), and placing u in P_3 implies no restriction.

The argument for $d = *, e = 1$ is similar. We have $\#M(G^o) = 2^n + \#I(G) + \#M(G)$ because the number of M-partitions with u in P_1 is the number of subsets of $V(G)$ (since P_3 must be empty). Similarly, the number of M-partitions with u in P_3 is the number of independent sets in G, and placing u to P_2 does not force any restriction and gives $\#M(G)$.

When $d = e = 1$, placing u to P_1 will make both P_2 and P_3 empty and gives at most one trivial partition. Also, placing u to P_2 (or P_3) will make P_1 empty, and thus, gives an instance of the independent set problem.

The only remaining cases are $d = 1, e = 0$ and $d = 0, e = 1$. Since $f = *$, these correspond to a special case of the modules from Theorem 3.4. According to Corollary 3.3, the number of such M-partitions with the first part non-empty is polynomial; on the other hand, the number of such M-partitions with the first part empty is #P-complete by Theorem 3.1. Since the total number of M-partitions is the sum of these two numbers, it must also be #P-complete.

Hence, in all cases, $\#I(G)$ can be reduced in polynomial time to $\#M(G)$. □

Lemma 3.4. *If $a = 0$, $b = f = *$, $c = 0$, then counting the number of M-partitions is #P-complete.*

Proof. Note that if $d = *$, then M will contain, as a principle submatrix, the matrix for independent sets and we conclude by Lemma 3.3. Also, $a = 0$ implies that at least one of d or e would be 1.

For all remaining cases, we reduce from the number of independent sets in a graph G. We consider two different cases separately.

If $d = 1$ and $e \neq 0$, then we use the universal vertex extension G^* of G. Observe that placing the universal vertex u in P_1 implies that all other vertices must be placed in P_2 or P_3 without any further restrictions. This coincides with the partitions of G according to the matrix for independent sets. Placing u in P_2 does not restrict vertices of G, because b, c, d,

and e are all from $\{1, *\}$, so it yields $\#M(G)$. Finally, mapping u to C forbids vertices of G from mapping to C (as $c = 0$), and this gives a polynomial time two by two case because $a = 0, d = 1$ (Theorem 3.1).

Otherwise, we either have $d = 1$, $e = 0$, or, $d = 0$, $e = 1$. This time, we use the isolated vertex extension G^o of G for our reduction. Notice that placing u in P_1 gives a tractable two by two sub-problem. When $d = 1$, placing u in P_2 restricts vertices of G from mapping to A and placing u in P_3 does not enforce any restrictions. When $e = 1$, placing u in P_3 forbids mapping vertices of G to A, while placing it in P_2 does not enforce any restrictions. In both cases, the number of M-partitions is $\#I(G) + \#M(G)$.

□

This completes the proof of Theorem 3.9.

3.5 New Results

In this Chapter, we studied the counting complexity of matrix partition problems. We completely classified time complexity of counting matrix partition problems for matrices of size less than 4. We showed that for any such partition matrix M, the problem of counting M-partition problems is either polynomial time solvable or # P-complete.

After our results were published in [69], Gobel, Goldberg, McQuillan, Richerby and Yamakami investigated one variant of this problem where every vertex of the input graph is equipped with a list of admissible parts [58]. This is called *counting list matrix partitions of graphs* and denoted by # List-M-partition. They determined the complexity of # List-M-partition problems. They reported that for every partition matrix M, the corresponding # List-M-partition problem is # P-complete or polynomial time computable.

Very recently, Dyer, Goldberg and Richerby extended our results for small matrices to matrices of size 4 [37]. They classified the counting complexity of M-partition problems (without lists) for matrices of size 4 by a computer-assisted proof and discovered that for every matrix M of size at most 4, the # M-partition and # List-M-partition problems are polynomially equivalent.

In the rest of this section, we discuss these new results in more detail.

3.5.1 Counting List Matrix Partitions of Graphs

Let M be a fixed partition matrix and denote by $\#$ List-M-partition the following problem.

Name: $\#$ List-M-partition
Input: A graph G together with lists $L(v)$ of admissible parts for every vertex v
Output: The number of M-partitions of G that respect lists L

Gobel, Goldberg, McQuillan, Richerby and Yamakami investigated this problem in [58]. They provided a polynomial time algorithm to solve tractable cases and proved that all the remaining cases are $\#$ P-complete.

Theorem 3.10. *[58] Let M be a partition matrix. Then $\#$ List-M is $\#$ P-complete or polynomial time computable.*

In fact, they studied a more general problem in which each list can only be selected from a predefined set of lists with specific conditions. Let M be a partition matrix of size m, and $\mathcal{L}$ be a family of sets over $[m]$. We say that $\mathcal{L}$ is *subset-closed* if for every set $A \in \mathcal{L}$, all subsets of A are also in $\mathcal{L}$. For fixed M and (fixed) subset-closed $\mathcal{L}$ they denoted by $\#\,\mathcal{L}$-M-Part the problem of counting list M-partitions of graphs in which every list of the input graph G is restricted to be selected from $\mathcal{L}$. It follows that *counting list matrix partition* problem is the special case of $\#\,\mathcal{L}$-M-Part with $\mathcal{L} = \mathcal{P}([m])$, i.e., there is no restriction on lists.

Theorem 3.11. *[58] Let M be a partition matrix and $\mathcal{L} \subseteq \mathcal{P}([m])$ be subset-closed. Then $\#\,\mathcal{L}$-M-Part is $\#$ P-complete or polynomial time computable.*

We have already mentioned in Section 2.7 that matrix partition problems generalize graph homomorphism problems. Indeed, when M has no 1s, it can be interpreted as the adjacency matrix of a graph by replacing $*$ with 1s, and the M-partition problem and all its variants, including problems with lists and counting problems, are equivalent to the corresponding (list or counting) graph homomorphism problem. Similarly, when M has no 0s, it can be interpreted as the adjacency matrix of the complement of a graph, by first converting all 1s to 0s, and then all $*$s to 1s. The authors in [58] generalized this concept as follows.

Let M be a partition matrix of size m and M' be a submatrix of M. The authors in [58] say that M' is *pure* when it has no 1s, or when it has no 0s. A family $\mathcal{L} \subseteq \mathcal{P}([m])$ is

M-purifying if for every $X, Y \in \mathcal{L}$, $M_{X,Y}$ is pure. For any set $\mathcal{L}$, its *subset closure* $\mathcal{S}(\mathcal{L})$ is the set defined below.

$$\mathcal{S}(\mathcal{L}) = \{X \mid \exists Y \in \mathcal{L} \text{ such that } X \subseteq Y\}.$$

Clearly, if $\mathcal{L}$ is an M-purifying set, then $\mathcal{S}(\mathcal{L})$ is also M-purifying.

For instance, while the matrix for clique cutset (depicted in Figure 2.2) is not a pure matrix itself, $\mathcal{L} = \{\{1,2\},\{2,3\}\}$ is a purifying set for it. As is its subset closure

$$\mathcal{S}(\mathcal{L}) = \{\{1\},\{2\},\{3\},\{1,2\},\{2,3\}\}.$$

A relation $R \subseteq A \times B$ is *rectangular* if for every $a, a' \in A$ and $b, b' \in B$, $(a,b),(a,b'),(a',b) \in R$ implies that (a',b') is also in R. Rectangular relations were first introduced by Bulatov and Dalmau in studying the complexity of counting constraint satisfaction problems [23]. It turns out that rectangular relations are also useful in determining the complexity of the $\#\,\mathcal{L}$-M-Part problems. For $X, Y \subseteq [m]$, a binary relation can be defined based on submatrix $M_{X,Y}$ as follows.

$$H^M_{X,Y} = \{(i,j) \in X \times Y | M_{i,j} = *\}.$$

A $\#\,\mathcal{L}$-M-*derectangularising sequence* is a sequence $\{D_i\}$ of k elements in $\mathcal{L}$ such that $D = \{D_1, D_2, \cdots, D_k\}$ is M-purifying and the relation H^M_D defined below is not rectangular.

$$H^M_D = H^M_{D_1,D_2} \circ H^M_{D_2,D_3} \circ \cdots H^M_{D_{k-1},D_k}.$$

Back to the clique cutset example, it is not hard to see that for $X = Y = \{1,2\}$, $H^M_{X,Y} = \{(1,1),(2,2)\}$ is a rectangular relation, while for $X = \{1,2\}$ and $Y = \{2,3\}$, $H^M_{X,Y} = \{(1,3),(2,2),(2,3)\}$ is not a rectangular relation. So, we have a $\#\,\mathcal{L}$-M-derectangularising sequence with $\mathcal{L} = \{\{1\},\{2\},\{3\},\{1,2\},\{2,3\}\}$ by choosing $k = 2$ and let $D_1 = \{1,2\}$ and $D_2 = \{2,3\}$.

The dichotomy condition for counting list matrix partitions of graphs can now be expressed using derectangularising sequences.

Theorem 3.12. *[58] Let M be a partition matrix and $\mathcal{L} \subseteq \mathcal{P}([m])$ be subset-closed. If there is an $\#\,\mathcal{L}$-M*derectangularising sequence then $\#\,\mathcal{L}$-M-Part *is $\#$P-complete. Otherwise, it is polynomial time computable.*

When $\mathcal{L}$ is M-purifying, they first reduce the problem to an instance of a related counting constraint satisfaction problem and then show that the CSP instance has a number of nice characteristics that makes it simple enough to be solved efficiently using standard arc-consistency techniques. In the case that $\mathcal{L}$ is not M-purifying, they develop data structures and algorithms to solve $\#\,\mathcal{L}$-M-Part using a series of $\#\,\mathcal{L}_{\mathrm{i}}$-M-Part problems in which every $\mathcal{L}_i$ is M-purifying. The data structures are based on a special case of sparse-dense partitions that were introduced in [50] and subcube decompositions of bipartite graphs.

They also studied the complexity of testing the dichotomy condition. They showed that for a given matrix M and $\mathcal{L} \subseteq \mathcal{P}([m])$ (as input), the problem of deciding whether there exists an M-derectangularising sequence in $\mathcal{L}$ is NP-complete.

3.5.2 Counting 4×4 Matrix Partitions of Graphs

In this part, we briefly review new results of Dyer, Goldberg and Richerby on counting partitions of graphs as published in [37]. For a formal definition of the problem we refer the reader to Chapter 3. Their main result is the following theorem.

Theorem 3.13. *[37] Let M be a partition matrix of size 4. Then, $\#$ M-partition is $\#$ P-complete if M has a derectangularising sequence, and is polynomial time computable otherwise.*

Theorem 3.13 implies that the counting complexity of M-partition problems coincides with the counting complexity of list M-partition problems for all matrices of size 4. In fact, they conjecture that this is true for every partition matrix.

Conjecture 3.1. *[37] Let M be any partition matrix. Then $\#$ M-partition and list $\#\,\mathcal{L}$-M-Part problems have the same complexity.*

To prove $\#$ P-completeness, the authors in [37] develop gadget-based techniques to reduce from $\#$ M$'$-partition problem when matrix M' is a principal submatrix of M with size less than 4. To find the appropriate matrix M' for every matrix M, they use a computer program which employs interpolation techniques with equations obtained from adding gadgets of different sizes and types to an arbitrary input graph G.

To show that the problem is polynomial time solvable, they prove that when a certain set of conditions is satisfied for a matrix M, it cannot have a derectangularising sequence,

and hence, counting M-partitions will be easy as a result of Theorem 3.12. Again, they use a computer search to find the matrices that meet these conditions.

These help them classify the counting complexity of M-partition problems for all but a small number of 4 by 4 matrices. Finally, they present individual proofs for each of the remaining cases.

Chapter 4

Approximation Algorithms

4.1 Introduction

We introduced the minimum cost homomorphism problem for graphs in Section 2.6. We mentioned there that certain minimum cost homomorphism problems have polynomial time algorithms [61, 64, 62, 116], but most are NP-hard. Therefore we investigate the approximability of these problems.

For a fixed bipartite graph H, it is known that when the list homomorphism problem to H is not tractable, then there is no polynomial time approximation algorithm (with a multiplicative guarantee) for the minimum cost graph homomorphism problem to H (see Theorem 2.8).

Theorem 4.1. *[105] Let H be a fixed bipartite graph. Unless* P = NP*, there is no polynomial time approximation algorithm for MinHOM(H) when H is not a proper interval bigraph.*

Indeed, the inference of the authors in [105] is valid for all (directed) graphs.

Theorem 4.2. *Let H be a fixed digraph. If the list homomorphism problem to H is* NP*-complete, then there is no polynomial time approximation algorithm for MinHOM(H), unless* P = NP*.*

Proof. We show how to solve the list homomorphism problem to H using an approximation algorithm for MinHOM(H). Let G, L_v ($v \in V(G)$) be an instance of the LHOM(H). Construct an instance of the MinHOM(H) problem with the same input graph G and the homomorphism cost function c defined below. For every $v \in V(G), a \in V(H)$, let

$c(v,a) = 0$ when $a \in L_v$ and $c(v,a) = 1$ otherwise. Observe that when there is a list homomorphism of G to H, then there is a homomorphism of G to H with cost 0. Otherwise, every homomorphism of G to H costs at least 1. Thus, any polynomial time approximation algorithm for MinHOM(H) with a multiplicative guarantee can be employed to solve any instance of the corresponding list homomorphism problem. □

In this Chapter, we prove that the converse is also true.

Theorem 4.3. *Let H be a fixed digraph. If the list homomorphism problem to H is polynomial time solvable, then there is a polynomial time approximation algorithm for the minimum cost graph homomorphism problem to H.*

Hence, we conclude that there is a dichotomy classification for the problems of approximately finding minimum cost graph homomorphisms to directed graphs, and interestingly, this classification coincides with the dichotomy classification for list homomorphism problems. In the process, we investigate the complexity of the bottleneck version of the minimum cost graph homomorphism problems.

The approximation ratio guaranteed by our general algorithm for minimum cost graph homomorphism problems is n, the number of vertices in the input graph G. We improve this ratio for the approximable cases where the target graph is a simple graph without loops. Theorems 2.3 and 4.3 imply that when H is not a co-circular arc bigraph, there is no polynomial time approximation algorithm for MinHOM(H) unless P = NP. We show that when H is a co-circular arc bigraph, there is a constant ratio approximation algorithm for MinHOM(H). For the purposes of the approximation, we use the characterization of co-circular arc bigraphs by the existence of min orderings (see Theorem 4.4 below). A *min ordering* of a graph H is an ordering of its vertices $a_1, a_2, \cdots, a_n$, such that the existence of the edges $a_ia_j, a_{i'}a_{j'}$ with $i < i', j' < j$ implies the existence of the edge $a_ia_{j'}$. For bigraphs, it is more convenient to speak of two orderings, and we define a *min ordering* of a bigraph H to be an ordering $a_1, a_2, \ldots, a_p$ of the white vertices and an ordering $b_1, b_2, \ldots, b_q$ of the black vertices, such that the existence of the edges $a_ib_j, a_{i'}b_{j'}$ with $i < i', j' < j$ implies the existence of the edge $a_ib_{j'}$; and a *min-max ordering* of a bigraph H to be an ordering of $a_1, a_2, \ldots, a_p$ of the white vertices and an ordering $b_1, b_2, \ldots, b_q$ of the black vertices, so that the existence of the edges $a_ib_j, a_{i'}b_{j'}$ with $i < i'$, $j' < j$ implies the existence of the edges $a_ib_{j'}, a_{i'}b_j$. (Both definitions are instances of a general definition of min ordering for directed graphs [79].)

Theorem 4.4. *[74] A bigraph H is a co-circular arc graph if and only if it admits a min ordering.*

We present a constant ratio approximation algorithm for MinHOM(H) when H is a bigraph that admits a min ordering. Our algorithm is based on the method of randomized rounding, a novel technique of randomized shifting, and then a derandomization process. The constant only depends on the structure of the fixed target graph H, and is bounded by the number of vertices in H.

In the realm of reflexive graphs, it is known that the class of interval graphs are precisely the reflexive graphs that admit a min ordering [77].

Theorem 4.5. *[77] A reflexive graph H is a proper interval graph if and only if it admits a min ordering.*

This enables us to provide an analogous constant ratio approximation algorithm for MinHOM(H) when H is any fixed interval graph.

The rest of this Chapter is organized as follows. In Section 4.2 we present our general approximation algorithm for minimum cost graph homomorphism problems and show that its worst case approximation ratio is at most n, the number of vertices in the input graph G. In Section 4.3 we discuss an integer linear program that computes an exact solution to MinHOM(H) when H has a min-max ordering. Recall that the authors in [62] used a minimum weighted cut problem equivalent to this linear program, to solve MinHOM(H) when H admits a min-max ordering. Our constant ratio approximation algorithms for (irreflexive) proper interval bigraphs and (reflexive) proper interval graphs are based on this integer linear program formulation. In Section 4.4 we present our novel constant ratio approximation algorithm for co-circular arc graphs with clique covering number two. In Section 4.5 we extend our results to reflexive graphs and show that there exists a $|V(H)|$-approximation algorithm when H is an interval graph. This algorithm is the analogue of our algorithm for irreflexive graphs.

4.2 Bottleneck Minimum Cost Graph Homomorphism Problems

Let H be a fixed digraph. The *bottleneck minimum cost graph homomorphism problem* to H is the optimization problem that, for an input graph G together with homomorphism

cost function $c : V(G) \times V(H) \to N$, asks for a homomorphism f that minimizes the maximum cost of mapping any vertex u in G to to its image $f(u)$, i.e., its goal is to minimize $max_{u \in V(G)} \{c(u, f(u))\}$. In this Section, we investigate the corresponding decision version, denoted BMinHom(H), which takes as input a graph G, a homomorphism cost function c and an integer k and asks if there is a homomorphism of G to H, such that $c(u, f(u)) \leq k$ for every vertex $u \in V(G)$. We refer to $max_{u \in V(G)} \{c(u, f(u))\}$ as the bottleneck cost of f.

We present a complete dichotomy for BMinHOM(H) problems that, again, coincides with the dichotomy of list homomorphism problems.

Theorem 4.6. *Let H be a fixed directed graph. The bottleneck minimum cost graph homomorphism problem to H and the list homomorphism problem to H are polynomially equivalent.*

Proof. First, we reduce BMinHOM(H) to LHOM(H). Let a digraph G together with a homomorphism cost function c and an integer k be an instance of BMinHOM(H). Construct an instance (G', L) with $G' = G$ and $L(v) = \{a \in V(H) | c(u, a) \leq k\}$. Observe that every homomorphism $f : V(G') \to V(H)$ with respect to the lists L is also a homomorphism of G to H with bottleneck cost at most k, and vice versa.

Now, we reduce LHOM(H) to BMinHOM(H). Given an instance (G, L) of LHOM(H), construct an instance (G', c, k) of BMinHOM(H) as follows. Again, let $G' = G$ and c be the binary function that represents the (complement of the) lists $L(u)$, i.e., let $c(u, a) = 0$ whenever $a \in L(u)$ and $c(u, a) = 1$ otherwise. Finally, let $k = 0$. Again, it is not hard to see that f is a homomorphism of G to H with respect to the lists L if and only if it is a homomorphism of G' to H with bottleneck cost 0. □

The rest of this Section is dedicated to the proof of Theorem 4.3. For the purposes of the approximation, we construct a sequence of instances of the corresponding bottleneck minimum cost graph homomorphism problem, in a particular order, until we find an instance with a **yes** answer, and then, we return that as the approximate solution.

Proof. Let H be any fixed digraph and let a digraph G together with a homomorphism cost function c be an instance of MinHOM(H). Denote by $R_c = \{c_0, c_1, \cdots, c_k\}$ the range of the function c, i.e., the set of all values $c(u, i)$ for every u and i. Without loss of generality, assume that $c_0 \leq c_1 \leq \cdots \leq c_k$.

For every $0 \leq i \leq k$, let $\mathcal{P}_i$ denote an instance of BMinHOM(H) with input graph G together with cost function c and bottleneck cost c_i. Let t be the smallest index such that P_t has a **yes** answer.

Notice that $k \leq |V(G)| \times |V(H)|$ is linear in size of G (since H is fixed), and hence, the algorithm generates at most a linear number of instances of the bottleneck minimum cost homomorphism problem. (Employing a binary search will in fact reduce the number of queries to LHOM(H) to at most $O(log(|V(G)|)$.) Moreover, finding R_c, sorting all its elements and constructing all the instances P_i can also be performed in polynomial time. So, as long as BMinHOM(H) can be solved in polynomial time, this algorithm also runs in polynomial time.

Claim: Every solution of $\mathcal{P}_t$ costs at most $|V(G)|$ times the minimum cost of any homomorphism of G to H.

Observe that if $t = 0$, then all lists consist of only one element, c_0, which is the smallest possible cost for all vertices, and hence, any solution of $\mathcal{P}_0$ is in fact an optimum solution. So, we may assume that $t > 0$. Thus, the answer to $\mathcal{P}_{t-1}$ is **no** and there is no homomorphism of G to H with bottleneck cost c_{t-1}. This implies that any homomorphism of G to H will cost at least c_t. On the other hand, any solution of $\mathcal{P}_t$ costs at most $|V(G)| \times c_t$.

This, together with Theorem 4.6, completes the proof of Theorem 4.3. □

4.3 An Exact Algorithm for Proper Interval Bigraphs

It is known that when H is a fixed bigraph with a min-max ordering, there is an exact algorithm for the problem MinHOM(H). In fact, the authors in [62] provided a transformation of this problem to a minimum weighted cut problem for every such graph H. In this Section, we discuss a linear program that is equivalent to the minimum weighted cut formulation discussed in [62].

Suppose H admits a min-max ordering, with the white vertices ordered $a_1, a_2, \cdots, a_p$, and the black vertices ordered $b_1, b_2, \cdots, b_q$. Define $\ell(i)$ to be the smallest subscript j such that b_j is a neighbour of a_i (and $\ell'(i)$ to be the smallest subscript j such that a_j is a neighbour of b_i) with respect to the ordering. Suppose G is a bigraph with white vertices u and black vertices v. We seek a minimum cost homomorphism of G to H that preserves colours, i.e., maps white vertices of G to white vertices of H and similarly for black vertices.

We define a set of variables $x_{u,i}, x_{v,j}$ for all vertices u and v of G and all $i = 1, 2, \ldots, p+$

$1, j = 1, 2, \ldots, q+1$, and the following linear system $\mathcal{S}$.
For all vertices u (respectively v) in G and $i = 1, \ldots, p$ (respectively $j = 1, \ldots, q$)

- $x_{u,i} \geq 0$ (respectively $x_{v,j} \geq 0$)
- $x_{u,1} = 1$ (respectively $x_{v,1} = 1$)
- $x_{u,p+1} = 0$ (respectively $x_{v,q+1} = 0$)
- $x_{u,i+1} \leq x_{u,i}$ (respectively $x_{v,j+1} \leq x_{v,j}$).

For all edges uv of G and $i = 1, 2, \ldots, p$, $j = 1, 2, \ldots, q$

- $x_{u,i} \leq x_{v,\ell(i)}$
- $x_{v,j} \leq x_{u,\ell'(j)}$

Theorem 4.7. *There is a one-to-one correspondence between homomorphisms of G to H and integer solutions of $\mathcal{S}$. Furthermore, the cost of the homomorphism is equal to $\sum_{u,i} c(u,i)(x_{u,i} - x_{u,i+1}) + \sum_{v,j} c(v,j)(x_{v,j} - x_{v,j+1})$.*

Proof. If $f : G \to H$ is a homomorphism, we set the value $x_{u,i} = 1$ if $f(u) = a_t$ for some $t \geq i$, otherwise we set $x_{u,i} = 0$; and similarly for $x_{v,j}$. Now all the variables are non-negative, we have all $x_{u,1} = 1$, $x_{u,p+1} = 0$, and $x_{u,i+1} \leq x_{u,i}$; and similarly for $x_{v,j}$. It remains to show that $x_{u,i} \leq x_{v,\ell(i)}$ for any edge uv of G and any subscript i. (The proof of $x_{v,j} \leq x_{u,\ell'(j)}$ is analogous.) Suppose for a contradiction that $x_{u,i} = 1$ and $x_{v,\ell(i)} = 0$, and let $f(u) = a_r$, $f(v) = b_s$. This implies that $x_{u,r} = 1, x_{u,r+1} = 0$, whence $i \leq r$; and that $x_{v,s} = 1$, whence $s < \ell(i)$. Since both $a_i b_{\ell(i)}, a_r b_s$ are edges of H, the fact that we have a min ordering implies that $a_i b_s$ must also be an edge of H, contradicting the definition of $\ell(i)$.

Conversely, if there is an integer solution for $\mathcal{S}$, we define a homomorphism f as follows: we let $f(u) = a_i$ when i is the largest subscript with $x_{u,i} = 1$ (and similarly, $f(v) = b_j$ when j is the largest subscript with $x_{v,j} = 1$). Clearly, every vertex of G is mapped to some vertex of H, of the same colour. We prove that this is indeed a homomorphism by showing that every edge of G is mapped to an edge of H. Let $e = uv$ be an edge of G, and assume $f(u) = a_r, f(v) = b_s$. We will show that $a_r b_s$ is an edge of H. Observe that $1 = x_{u,r} \leq x_{v,\ell(r)} \leq 1$ and $1 = x_{v,s} \leq x_{u,\ell'(s)} \leq 1$, so we must have $x_{u,\ell'(s)} = x_{v,\ell(r)} = 1$.

Also observe that $x_{u,i} = 0$ for all $i > r$, and $x_{v,j} = 0$ for all $j > s$. Thus, $\ell(r) \leq s$ and $\ell'(s) \leq r$. Since $a_r b_{\ell(r)}$ and $a_{\ell'(s)} b_s$ are edges in H, we must have the edge $a_r b_s$, as we have a min-max ordering.

Furthermore, $f(u) = a_i$ if and only if $x_{u,i} = 1$ and $x_{u,i+1} = 0$, so, $c(u,i)$ contributes to the sum if and only if $f(u) = a_i$ (and similarly, if $f(v) = b_j$). □

We have translated the minimum cost homomorphism problem to an integer linear program: minimize the objective function in Theorem 4.7 over the linear system $\mathcal{S}$. In fact, this integer linear program corresponds to a minimum cut problem in an auxiliary network, and can be solved by network flow algorithms [62, 105]. We shall enhance the above system $\mathcal{S}$ to obtain an approximation algorithm for the case H is only assumed to have a min ordering.

4.4 An Approximation Algorithm for Co-Circular Arc Bigraphs

In this section we describe our approximation algorithm for MinHOM(H) in the case the fixed bigraph H has a min ordering, i.e., is a co-circular arc bigraph, cf. Theorem 4.4. We recall that if H is not a co-circular arc bigraph, then the list homomorphism problem ListHOM(H) is NP-complete [46], and this implies that MinHOM(H) is not approximable for such graphs H [105]. By Theorem 4.4 we conclude the following.

Theorem 4.8. *If a bigraph H has no min ordering, then MinHOM(H) is not approximable.*

Our main result is the following converse: if H has a min ordering (is a co-circular arc bigraph), then there exists a constant ratio approximation algorithm. (Since H is fixed, $|V(H)|$ is a constant.)

Theorem 4.9. *If H is a bigraph that admits a min ordering, then MinHOM(H) has a $|V(H)|$-approximation algorithm.*

Proof. Suppose H has a min ordering with the white vertices ordered $a_1, a_2, \cdots, a_p$, and the black vertices ordered $b_1, b_2, \cdots, b_q$. Let E' denote the set of all pairs $a_i b_j$ such that $a_i b_j$ is not an edge of H, but there is an edge $a_i b_{j'}$ of H with $j' < j$ and an edge $a_{i'} b_j$ of H with $i' < i$. Let $E = E(H)$ and define H' to be the graph with vertex set $V(H)$ and edge set $E \cup E'$. (Note that E and E' are disjoint sets.)

Observation 1. The ordering $a_1, a_2, \cdots, a_p$, and $b_1, b_2, \cdots, b_q$ is a min-max ordering of H'.

We show that for every pair of edges $e = a_i b_{j'}$ and $e' = a_{i'} b_j$ in $E \cup E'$, with $i' < i$ and $j' < j$, both $f = a_i b_j$ and $f' = a_{i'} b_{j'}$ are in $E \cup E'$.

If both e and e' are in E, $f \in E \cup E'$ and $f' \in E$.

If one of the edges, say e, is in E', there is a vertex $b_{j''}$ with $a_i b_{j''} \in E$ and $j'' < j'$, and a vertex $a_{i''}$ with $a_{i''} b_{j'} \in E$ and $i'' < i$. Now, $a_{i'} b_j$ and $a_i b_{j''}$ are both in E, so $f \in E \cup E'$. We may assume that $i'' \neq i'$, otherwise $f' = a_{i''} b_{j'} \in E$. If $i'' < i'$, then $f' \in E \cup E'$ because $a_{i'} b_{j''} \in E$; and if $i'' > i'$, then $f' \in E$ because $a_{i'} b_j \in E$.

If both edges e, e' are in E', then the earlier neighbours of a_i and b_j in E imply that $f \in E \cup E'$, and the earlier neighbours of $a_{i'}$ and $b_{j'}$ in E imply that $f' \in E \cup E'$.

Observation 2. Let $e = a_i b_j \in E'$. Then a_i is not adjacent in E to any vertex after b_j, or b_j is not adjacent in E to any vertex after a_i.

This easily follows from the fact that we have a min ordering.

Our algorithm first constructs the graph H' and then proceeds as follows. Consider an input bigraph G. Since H' has a min-max ordering, we can form the system $\mathcal{S}$ of linear inequalities for H'. By Theorem 4.7, homomorphisms of G to H' are in a one-to-one correspondence with integer solutions of $\mathcal{S}$. However, we are interested in homomorphisms of G to H, not H'. Therefore we shall add further inequalities to $\mathcal{S}$ to ensure that we only admit homomorphisms of G to H, i.e., avoid mapping edges of G to the edges in E'.

For every edge $e = a_i b_j \in E'$ and every edge $uv \in E(G)$, two of the following inequalities will be added to $\mathcal{S}$.

- if a_s is the first neighbour of b_j after a_i, we add the inequality

  $$x_{v,j} \leq x_{u,s} + \sum_{a_t b_j \in E,\ t<i} (x_{u,t} - x_{u,t+1})$$

- else if b_j has no neighbours after a_i, we add the inequality

  $$x_{v,j} \leq x_{v,j+1} + \sum_{a_t b_j \in E,\ t<i} (x_{u,t} - x_{u,t+1})$$

- if b_s is the first neighbour of a_i after b_j, we add the inequality

  $$x_{u,i} \leq x_{v,s} + \sum_{a_i b_t \in E,\ t<j} (x_{v,t} - x_{v,t+1})$$

- else if a_i has no neighbour after b_j, we add the inequality

  $$x_{u,i} \leq x_{u,i+1} + \sum_{a_i b_t \in E,\ t<j} (x_{v,t} - x_{v,t+1}).$$

Claim: There is a one-to-one correspondence between homomorphisms of G to H and integer solutions of the expanded system $\mathcal{S}$.

The correspondence between the integer solutions and the homomorphisms is defined as before. Thus we have a homomorphism of G to H' if and only if the old inequalities are satisfied. We shall show that the additional inequalities are also satisfied if and only if each edge of G is mapped to an edge in E, i.e., we have a homomorphism to H.

Suppose f is a homomorphism of G to H', obtained from an integer solution for $\mathcal{S}$, and, for some edge uv of G, let $f(u) = a_i, f(v) = b_j$ (recall that $a_ib_j \in E'$). We have $x_{u,i} = 1$, $x_{u,i+1} = 0$, $x_{v,j} = 1$, $x_{v,j+1} = 0$, and for all $a_tb_j \in E$ with $t < i$ we have $x_{u,t} - x_{u,t+1} = 0$. If a_s is the first neighbour of b_j after a_i, then we will also have $x_{u,s} = 0$, and so the first inequality fails. Else if b_j is not adjacent to any vertex after a_i, and the second inequality fails. The remaining two other cases are similar.

Conversely, suppose f is a homomorphism of G to H (i.e., f maps the edges of G to the edges in E). For a contradiction, assume that the first inequalities fails (the other inequalities are similar). This means that for some edge $uv \in E(G)$ and some edge $a_ib_j \in E'$, we have $x_{v,j} = 1$, $x_{u,s} = 0$, and the sum of $(x_{u,t} - x_{u,t+1}) = 0$, summed over all $t < i$ such that a_t is a neighbour of b_j. The latter two facts easily imply that $f(u) = a_i$. Since b_j has a neighbour after a_i, Observation 2 tells us that a_i has no neighbours after b_j, whence $f(v) = b_j$ and thus $a_ib_j \in E$, contradicting the fact that $a_ib_j \in E'$. This proves the Claim.

At this point, our algorithm will minimize the cost function over $\mathcal{S}$ in polynomial time using a linear programming algorithm. This will generally result in a fractional solution. (Even though the original system $\mathcal{S}$ is known to be totally unimodular [105] and hence have integral optima, we have added inequalities, and hence lost this advantage.) We will obtain an integer solution by a randomized procedure called *rounding*. We choose $X \in [0, 1]$, uniformly at random, and define the rounded values $x'_{u,i} = 1$ when $x_{u,i} \geq X$, and $x'_{u,i} = 0$ otherwise; and similarly for $x'_{v,j}$. It is easy to check that the rounded values satisfy the original inequalities, i.e., correspond to a homomorphism f of G to H'.

Now the algorithm will once more modify the solution f to become a homomorphism of G to H, i.e., to avoid mapping edges of G to the edges in E'. This will be accomplished by another randomized procedure, which we call *shifting*. We choose another variable $Y \in [0, 1]$, again, uniformly at random, which will guide the shifting. Let F denote the set of all edges in E' to which some edge of G is mapped by f. If F is empty, we need no shifting. Otherwise, let a_ib_j be an edge of F with maximum sum $i + j$ (among all edges of F). By

the maximality of $i+j$, we know that a_ib_j is the last edge of F from both a_i and b_j. Since $F \subseteq E'$, Observation 2 implies that $e = a_ib_j$ is also the last edge of E' from a_i or from b_j. Suppose e is the last edge of E from a_i. (The shifting process is similar in the other case.) So a_i does not have any edges of F or of E after a_ib_j. (There could be edges of $E'-F$, but since no edge of G is mapped to such edges, they don't matter.) We now consider, one by one, vertices u in G such that $f(u) = a_i$ and u has a neighbour v in G with $f(v) = b_j$. (Such vertices u exist by the definition of F.) For such a vertex u, consider the set of all vertices a_t with $t < i$ such that $a_tb_j \in E$. This set is not empty, since e is in E' because of two edges of E. Suppose the set consists of a_t with subscripts t ordered as $t_1 < t_2 < \ldots t_k$. The algorithm now selects one vertex from this set as follows. Let $P_{u,t} = \frac{x_{u,t}-x_{u,t+1}}{P_u}$, where

$$P_u = \sum_{a_tb_j \in E,\ t<i} (x_{u,t} - x_{u,t+1}).$$

Then a_{t_q} is selected if $\sum_{p=1}^{q} P_{u,t_p} < Y \leq \sum_{p=1}^{q+1} P_{u,t_p}$. Thus a concrete a_t is selected with probability $P_{u,t}$, which is proportional to the difference of the fractional values $x_{u,t} - x_{u,t+1}$.

When the selected vertex is a_t, we shift the image of the vertex u from a_i to a_t. This modifies the homomorphism f, and hence the corresponding values of the variables. Namely, $x'_{u,t+1}, \ldots, x'_{u,i}$ are reset to 0, keeping all other values the same. Note that these modified values still satisfy the original constraints, i.e., the modified mapping is still a homomorphism.

We repeat the same process for the next u with these properties, until a_ib_j is no longer in F (because no edge of G maps to it). This ends the iteration on a_ib_j, and we proceed to the next edge $a_{i'}b_{j'}$ with the maximum $i'+j'$ for the next iteration. Each iteration involves at most $|V(G)|$ shifts. After at most $|E'|$ iterations, the set F is empty and we no longer need to shift.

We now claim that because of the randomization, the cost of this homomorphism is at most $|V(H)|$ times the minimum cost of a homomorphism. We denote by w the value of the objective function with the fractional optimum $x_{u,i}, x_{v,j}$, and by w' the value of the objective function with the final values $x'_{u,i}, x'_{v,j}$, after the rounding and all the shifting. We also denote by w^* the minimum cost of a homomorphism of G to H. Obviously we have $w \leq w^* \leq w'$.

We now show that the expected value of w' is at most a constant times w. We focus

on the contribution of one summand, say $x'_{u,t} - x'_{u,t+1}$, to the calculation of the cost. (The other case, $x'_{v,s} - x'_{v,s+1}$, is similar.)

In any integer solution, $x'_{u,t} - x'_{u,t+1}$ is either 0 or 1. The probability that $x'_{u,t} - x'_{u,t+1}$ contributes to w' is the probability of the event that $x'_{u,t} = 1$ and $x'_{u,t+1} = 0$. This can happen in the following situations.

1. u is mapped to a_t by rounding, and is not shifted away. In other words, we have $x'_{u,t} = 1$ and $x'_{u,t+1} = 0$ after rounding, and these values don't change by shifting.

2. u is first mapped to some $a_i, i > t$, by rounding, and then re-mapped to a_t by shifting. This happens if there exist j and v such that uv is an edge of G mapped to $a_ib_j \in F$, and then the image of u is shifted to a_t, where $a_tb_j \in E$. In other words, we have $x'_{u,i} = x'_{v,j} = 1$ and $x'_{u,i+1} = x'_{v,j+1} = 0$ after rounding; and then u is shifted from a_i to a_t.

For the situation in 1, we compute the expectation as follows. The values $x'_{u,t} = 1$, $x'_{u,t+1} = 0$ are obtained by rounding if $x_{u,t+1} < X \leq x_{u,t}$, i.e., with probability $x_{u,t} - x_{u,t+1}$. The probability that they are not changed by shifting is at most 1, whence this situation occurs with probability at most $x_{u,t} - x_{u,t+1}$, and the expected contribution is at most $c(u,t)(x_{u,t} - x_{u,t+1})$.

For the situation in 2, we first compute the contribution for a fixed i (for which there exist j and v as described above). The values $x'_{u,i} = x'_{v,j} = 1$ and $x'_{u,i+1} = x'_{v,j+1} = 0$ are obtained by rounding if X satisfies $\max\{x_{u,i+1}, x_{v,j+1}\} < X \leq \min\{x_{u,i}, x_{v,j}\}$, i.e., with probability $\min\{x_{u,i}, x_{v,j}\} - \max\{x_{u,i+1}, x_{v,j+1}\} \leq x_{v,j} - x_{u,i+1} \leq x_{v,j} - x_{u,s} \leq P_u$. In the last two inequalities above we have assumed that a_s is the first neighbour of b_j after a_i, and used the first inequality added above the Claim. If b_j has no neighbours after a_i, the proof is analogous, using the second added inequality. When uv maps to a_ib_j, we shift u to a_t with probability $P_{u,t} = \frac{(x_{u,t} - x_{u,t+1})}{P_u}$, so the overall probability is also at most $x_{u,t} - x_{u,t+1}$, and the expected contribution for a fixed i (with its j and v) is also at most $c(u,t)(x_{u,t} - x_{u,t+1})$.

Let r denote the number of vertices of H, of the same colour as a_t, that are incident with some edges of E'. Clearly the situation in 2 can occur at for at most r different values of i. Therefore a fixed u in G contributes at most $(1+r)c(u,t)(x_{u,t} - x_{u,t+1})$ to the expected

value of w'. Thus the expected value of w' is at most

$$(1+r)\,(\sum_{u,i} c(u,i)(x_{u,i}-x_{u,i+1}) + \sum_{v,j} c(v,j)(x_{v,j}-x_{v,j+1})) \leq (1+r)w.$$

Since we have $w \leq w^*$, this means that the expected value of w' is at most $(1+r)w^*$. Note that $1+r \leq 1+|E'|$, and also $1+r < |V(H)|$ because a_1 (and b_1) are not incident with any edges of E' by definition.

At this point we have proved that our two-phase randomized procedure produces a homomorphism whose expected cost is at most $(1+r)$ times the minimum cost. It can be transformed to a deterministic algorithm as follows. There are only polynomially many values $x_{u,t}$ (at most $|V(G)|\dot{|}V(H)|$). When X lies anywhere between two such consecutive values, all computations will remain the same. Thus we can derandomize the first phase by trying all these values of X and choosing the best solution. Similarly, there are only polynomially many values of the partial sums $\sum_{p=1}^{q} P_{u,t_p}$ (again at most $|V(G)|\dot{|}V(H)|$), and when Y lies between two such consecutive values, all computations remain the same. Thus we can also derandomize the second phase by trying all possible values and choosing the best. Since the expected value is at most $(1+r)$ times the minimum cost, this bound also applies to this best solution. □

Corollary 4.1. *Let H be a co-circular arc bigraph in which at most r vertices of either colour are incident to edges of E', and let $c \geq 1+r$ be any constant.*

Then the problem MinHOM(H) has a c-approximation algorithm.

Note that c can be taken to be $|V(H)|$, or $1+|E'|$, as noted above. For $c = 1+|E'|$, we have an approximation with best bound when E' is small, in particular, an exact algorithm when E' is empty.

Finally, we conclude the following classification for the complexity of approximation of minimum cost homomorphism problems for irreflexive graphs.

Corollary 4.2. *Let H be an irreflexive graph.*

Then the problem MinHOM(H) has a polynomial time constant ratio approximation algorithm if H is a co-circular arc bigraph, and is not approximable otherwise.

4.5 An Approximation Algorithm for Interval Graphs

Interestingly, all our steps can be repeated verbatim for the case of reflexive graphs. If H is a reflexive graph with min-max ordering $a_1, a_2, \ldots, a_n$, we again define $\ell(i)$ as the smallest subscript j, with respect to this ordering, such that a_j is a neighbour of a_i. For an input graph G, we again define variables $x_{u,i}$, for $u \in V(G), i = 1, 2, \ldots, n$, and the same system of linear inequalities $\mathcal{S}$ (restricted to the u's), and obtain an analogue of Theorem 4.7. Provided H has a min ordering, we can again add edges E' as above to produce a reflexive graph H' with a min-max ordering, with the analogous properties expressed in Observations 1 and 2. We can add the corresponding inequalities to $\mathcal{S}$ as above, and there will again be a one-to-one correspondence between homomorphisms of G to H and the integer solutions to the system. Finally, we can define the approximation by the same sequence of rounding and shifting. Everything works exactly as before because it only depends on the definition of min (and min-max) ordering, which are the same. Finally, we use the fact that a reflexive graph has a min ordering if and only if it is an interval graph [43, 77], and the fact that the list homomorphism problem ListHOM(H) is NP-complete if the reflexive graph H is not an interval graph [43]. The last facts implies, as in [105], that the problem MinHOM(H) is not approximable if H is a reflexive graph that is not an interval graph.

Theorem 4.10. *Let H be a reflexive graph.*

The problem MinHOM(H) has a polynomial time $|V(H)|$-approximation algorithm if H is an interval graph, and is not approximable otherwise.

Before we proceed to the proof of Theorem 4.10, we shall give an analogous system of linear equations to find the exact solution of MinHOM(H) for reflexive graphs H that admit a min-max ordering.

Again, we define a set of variables $x_{u,i}$ for each vertex u of G and all $i = 1, 2, \ldots, n, n+1$, and the following linear system $\mathcal{S}$.
For every vertex u in G and all $i = 1, \ldots, n, n+1$

- $x_{u,i} \geq 0$
- $x_{u,1} = 1$
- $x_{u,n+1} = 0$
- $x_{u,i+1} \leq x_{u,i}$.

For all edges uv of G and all $i = 1, 2, \ldots, n$

- $x_{u,i} \leq x_{v,\ell(i)}$

Theorem 4.11. *There is a one-to-one correspondence between homomorphisms of G to H and integer solutions of $\mathcal{S}$. Furthermore, the cost of the homomorphism is equal to* $\sum_{u,i} c(u,i)(x_{u,i} - x_{u,i+1})$.

Proof. If $f : G \to H$ is a homomorphism, we set the value $x_{u,i} = 1$ if $f(u) = a_t$ for some $t \geq i$, otherwise we set $x_{u,i} = 0$. We show that $x_{u,i} \leq x_{v,\ell(i)}$ for any edge uv of G and any subscript i. Observing that other equations are satisfied is straight forward. Suppose for a contradiction that $x_{u,i} = 1$ and $x_{v,\ell(i)} = 0$, and let $f(u) = a_r$, $f(v) = a_s$. This implies that $x_{u,r} = 1, x_{u,r+1} = 0$, whence $i \leq r$; and that $x_{v,s} = 1$, whence $s < \ell(i)$. Since both $a_i a_{\ell(i)}, a_r a_s$ are edges of H, the fact that we have a min ordering implies that $a_i a_s$ must also be an edge of H, contradicting the definition of $\ell(i)$.

Conversely, if there is an integer solution for $\mathcal{S}$, we define a homomorphism f as follows: we let $f(u) = a_i$ when i is the largest subscript with $x_{u,i} = 1$. Clearly, every vertex of G is mapped to some vertex of H, We prove that this is indeed a homomorphism by showing that every edge of G is mapped to an edge of H. Let $e = uv$ be an edge of G, and assume $f(u) = a_r, f(v) = a_s$. We will show that $a_r a_s$ is an edge of H. Observe that $1 = x_{u,r} \leq x_{v,\ell(r)} \leq 1$ and $1 = x_{v,s} \leq x_{u,\ell(s)} \leq 1$, so we must have $x_{u,\ell(s)} = x_{v,\ell(r)} = 1$. Also observe that $x_{u,i} = 0$ for all $i > r$, and $x_{v,j} = 0$ for all $j > s$. Thus, $\ell(r) \leq s$ and $\ell(s) \leq r$. Since $a_r a_{\ell(r)}$ and $a_{\ell(s)} a_s$ are edges in H, we must have the edge $a_r a_s$, as we have a min-max ordering.

Furthermore, $f(u) = a_i$ if and only if $x_{u,i} = 1$ and $x_{u,i+1} = 0$, so, $c(u,i)$ contributes to the sum if and only if $f(u) = a_i$. □

Now, we can explain the proof of Theorem 4.10.

Proof. Suppose H has a min ordering with the vertices ordered $a_1, a_2, \cdots, a_n$. Let E' denote the set of all pairs $a_i a_j$ such that $a_i a_j$ is not an edge of H, but there is an edge $a_i a_{j'}$ of H with $j' < j$ and an edge $a_{i'} a_j$ of H with $i' < i$. Let $E = E(H)$ and define H' to be the graph with vertex set $V(H)$ and edge set $E \cup E'$.

Similar to the case of bipartite graphs, it is easy to note the following facts.

Observation 1. The ordering $a_1, a_2, \cdots, a_n$ is a min-max ordering of H'.

Observation 2. Let $e = a_i a_j \in E'$. Then a_i is not adjacent in E to any vertex after a_j, or a_j is not adjacent in E to any vertex after a_i.

Our algorithm once again constructs the graph H' and then proceeds by forming the system $\mathcal{S}$ of linear inequalities for H'. By Theorem 4.11, homomorphisms of G to H' are in a one-to-one correspondence with integer solutions of $\mathcal{S}$. Similar to the case of irreflexive graphs, we shall add further inequalities to $\mathcal{S}$ to ensure that we avoid mapping edges of G to the edges in E'.

For every edge $e = a_i a_j \in E'$ and every edge $uv \in E(G)$, we add one of the following inequalities to $\mathcal{S}$.

- if a_s is the first neighbour of a_j after a_i, we add the inequality

  $$x_{v,j} \leq x_{u,s} + \sum_{a_t a_j \in E,\ t<i} (x_{u,t} - x_{u,t+1})$$

- else if a_j has no neighbours after a_i, we add the inequality

  $$x_{v,j} \leq x_{v,j+1} + \sum_{a_t a_j \in E,\ t<i} (x_{u,t} - x_{u,t+1})$$

An analogous deduction can be used to show that there is again a one-to-one correspondence between homomorphisms of G to H and integer solutions of the expanded system $\mathcal{S}$, where the correspondence between the integer solutions and the homomorphisms is defined as before.

At this point, our algorithm will minimize the cost function over $\mathcal{S}$ in polynomial time using a linear programming algorithm. This will generally result in a fractional solution. We will obtain an integer solution by running exactly the same randomized rounding and shifting procedures discussed earlier for irreflexive graphs, guided by random variables $X, Y \in [0, 1]$, respectively. When shifting vertices with edges mapped to an edge $a_i a_j$, we use the same probabilities $P_{u,t} = \frac{x_{u,t} - x_{u,t+1}}{P_u}$, where

$$P_u = \sum_{a_t a_j \in E,\ t<i} (x_{u,t} - x_{u,t+1})$$

and select a particular target a_{t_q} when $\sum_{p=1}^{q} P_{u,t_p} < Y \leq \sum_{p=1}^{q+1} P_{u,t_p}$. This gives an integer solution that satisfies the extended system $\mathcal{S}$ and corresponds to a homomorphism f of G to H.

We now claim that because of the randomization, the cost of this homomorphism is at most $|V(H)|$ times the minimum cost of a homomorphism. We denote by w the value of the objective function with the fractional optimum $x_{u,i}$, and by w' the value of the objective function with the final values $x'_{u,i}$, after the rounding and all the shifting. We also denote by w^* the minimum cost of a homomorphism of G to H. Obviously we have $w \leq w^* \leq w'$.

We now show that the expected value of w' is at most a constant times w. We focus on the contribution of one summand, say $x'_{u,t} - x'_{u,t+1}$, to the calculation of the cost.

In any integer solution, $x'_{u,t} - x'_{u,t+1}$ is either 0 or 1. The probability that $x'_{u,t} - x'_{u,t+1}$ contributes to w' is the probability of the event that $x'_{u,t} = 1$ and $x'_{u,t+1} = 0$. This can happen in the following situations.

1. u is mapped to a_t by rounding, and is not shifted away. In other words, we have $x'_{u,t} = 1$ and $x'_{u,t+1} = 0$ after rounding, and these values don't change by shifting.

2. u is first mapped to some $a_i, i > t$, by rounding, and then re-mapped to a_t by shifting. This happens if there exist j and v such that uv is an edge of G mapped to $a_i a_j \in E'$, and then the image of u is shifted to a_t, where $a_t a_j \in E$. In other words, we have $x'_{u,i} = x'_{v,j} = 1$ and $x'_{u,i+1} = x'_{v,j+1} = 0$ after rounding; and then u is shifted from a_i to a_t.

For the situation in 1, we compute the expectation as follows. The values $x'_{u,t} = 1$, $x'_{u,t+1} = 0$ are obtained by rounding if $x_{u,t+1} < X \leq x_{u,t}$, i.e., with probability $x_{u,t} - x_{u,t+1}$. The probability that they are not changed by shifting is at most 1, whence this situation occurs with probability at most $x_{u,t} - x_{u,t+1}$, and the expected contribution is at most $c(u,t)(x_{u,t} - x_{u,t+1})$.

For the situation in 2, we first compute the contribution for a fixed i (for which there exist j and v as described above). The values $x'_{u,i} = x'_{v,j} = 1$ and $x'_{u,i+1} = x'_{v,j+1} = 0$ are obtained by rounding if X satisfies $\max\{x_{u,i+1}, x_{v,j+1}\} < X \leq \min\{x_{u,i}, x_{v,j}\}$, i.e., with probability $\min\{x_{u,i}, x_{v,j}\} - \max\{x_{u,i+1}, x_{v,j+1}\} \leq x_{v,j} - x_{u,i+1} \leq x_{v,j} - x_{u,s} \leq P_u$. In the last two inequalities above we have assumed that a_s is the first neighbour of b_j after a_i, and used the first inequality added above the Claim. If b_j has no neighbours after a_i, the proof is analogous, using the second added inequality. When uv maps to $a_i b_j$, we shift u to a_t with probability $P_{u,t} = \frac{(x_{u,t} - x_{u,t+1})}{P_u}$, so the overall probability is also at most $x_{u,t} - x_{u,t+1}$, and the expected contribution for a fixed i (with its j and v) is also at most $c(u,t)(x_{u,t} - x_{u,t+1})$.

Let r denote the number of vertices of H that are incident with some edges of E'. Clearly the situation in 2 can occur at for at most r different values of i. Therefore a fixed u in G contributes at most $(1+r)c(u,t)(x_{u,t} - x_{u,t+1})$ to the expected value of w'. Thus the expected value of w' is at most

$$(1+r)\,(\sum_{u,i} c(u,i)(x_{u,i} - x_{u,i+1}) \leq (1+r)w.$$

Since we have $w \leq w^*$, this means that the expected value of w' is at most $(1+r)w^*$.

At this point we have proved that our two-phase randomized procedure produces a homomorphism whose expected cost is at most $(1+r)$ times the minimum cost. It can be transformed to a deterministic algorithm as follows. There are only polynomially many values $x_{u,t}$ (at most $|V(G)||V(H)|$). When X lies anywhere between two such consecutive values, all computations will remain the same. Thus we can derandomize the first phase by trying all these values of X and choosing the best solution. Similarly, there are only polynomially many values of the partial sums $\sum_{p=1}^{q} P_{u,t_p}$ (again at most $|V(G)||V(H)|$), and when Y lies between two such consecutive values, all computations remain the same. Thus we can also derandomize the second phase by trying all possible values and choosing the best. Since the expected value is at most $(1+r)$ times the minimum cost, this bound also applies to this best solution. □

Chapter 5

Graph Homomorphisms with Constrained Costs

5.1 Introduction

The minimum cost graph homomorphism problem MinHOM(H) has a fixed target graph H and its input is a graph G together with a homomorphism cost function $c : V(G) \times V(H) \to \mathbb{N}$ (with $c(u, i)$ being the cost of mapping a vertex $u \in V(G)$ to a vertex $i \in V(H)$), and the goal is to find a homomorphism f of G to H that minimizes $\sum_{u \in V(H)} c(u, f(u))$.

In this Chapter, we study the minimum cost graph homomorphism problem with *constrained costs*. Recall from Chapter 2 that for each instance of MinHOM(H), the function $c_i : V(G) \to \mathbb{N}$ denotes the function that gives the cost of mapping vertices of G to a specific vertex $i \in V(H)$, thus, $c_i(u) = c(u, i)$ for every vertex $u \in V(G)$. We consider instances of the problem for which the input meets the following criteria. For every vertex $i \in V(H)$, the cost of mapping all vertices of G to i is the same, that is, c_i is a constant function for every vertex $i \in V(H)$. Nevertheless, we still assume that these costs are part of the input (G, c).

There is a broad literature on similar problems in graph colouring. In particular, the minimum colour sum problem and the optimum cost chromatic partition problem discussed in Section 2.5 are both special cases of the minimum constrained cost graph homomorphism problem that we study in this chapter. Indeed, one can transform an instance of these problems into an instance of the minimum cost homomorphism problem by choosing the

complete graph K_n as the target graph (where n is the number of vertices in input graph G), and appropriate costs.

In the original problem when there are no restrictions on the values that the cost function can take, the list homomorphism problem sits nicely between the homomorphism problem and the minimum cost homomorphism problem. It generalizes the graph homomorphism problem by introducing lists of admissible vertices, while it can still be transformed conveniently into an instance of the MinHOM(H) using boolean cost values to simulate the constraints enforced by lists. On the contrary, there is no trivial way to transform an instance of a list homomorphism problem into an instance of the minimum constrained cost graph homomorphism problem.

We now give the formal definition of the problem. Let H be a fixed graph. The *minimum constrained cost graph homomorphism problem* for H is the optimization problem that asks for the minimum cost of any homomorphism of an input graph G together with a homomorphism cost function $c : V(H) \to \mathbb{N}$ to H. The cost of a homomorphism function $f : V(G) \to V(H)$ is defined exactly as before, that is, $cost(f) = \sum_{u \in V(G)} c(f(u))$. In this Chapter, we consider the standard decision version of this problem, that receives an additional input parameter k, and asks whether there is a homomorphism of G to H which costs at most k.

Our main result is a partial dichotomy classification for this problem. We prove that when H is not a chordal bipartite graph, then the minimum constrained cost homomorphism problem to H is NP-complete. On the other hand, when H is a proper interval bigraph, the problem is in P, as we can simply employ the polynomial algorithm for MinHOM(H) (see Section 4.3) to solve the problem with constrained costs efficiently.

Theorem 5.1. *Let H be a fixed graph. The minimum constrained cost graph homomorphism problem to H is* NP*-complete when H is not a chordal bipartite graph; and is in* P *when H is a proper interval bigraph.*

We also present a complete dichotomy classification for all target graphs H that are trees. We prove that when H is a tree that is not a proper interval bigraph, the problem remains NP-complete. Hence, we conclude that the constraints we consider do not impact the complexity of the minimum cost homomorphism problem.

Theorem 5.2. *Let H be a fixed tree. The minimum constrained cost graph homomorphism problem to H is* NP*-complete when H is not proper interval bigraph; and is in* P *otherwise.*

We assume that both the input graph G and the target graph H are connected. Observe that, similar to the original problem without constrained costs, when G is disconnected, the minimum cost for every connected component of G is an independent sub-problem that can be solved individually, and when H is disconnected, every connected component of G must map to a component of H that gives the minimum cost for that component.

The reductions we present in the proofs of Theorems 5.1 and 5.2 work on a more general version of the problem that allows positive integer weights $w(v)$ on the vertices of the input graph G and asks for a homomorphism that minimizes the weighted sum $\sum_{u \in V(H)} w(u) \times c(u, f(u))$. As we shall see in Section 5.2, adding vertex weights does not affect the time complexity of the problem and will allow us shorthand the gadgets we construct in our proofs.

The rest of this chapter is organized as follows. In Section 5.2, we formally introduce the minimum cost graph homomorphism problem with vertex weights and show that when the weights are positive integers bounded by a polynomial in the size of the input graph G, the time complexity of the problem remains the same. Then, we prove Theorem 5.1 in the following two sections. Specifically, we show in Section 5.3 that the (weighted) minimum constrained cost graph homomorphism problem is NP-complete when the target graph H contains a hexagon as an induced subgraph, and in Section 5.4 when H contains an even cycle of length at least 8. Finally, we prove Theorem 5.2 in Section 5.5.

5.2 Vertex Weights and Minimum Cost Graph Homomorphism Problems

In this section, we study the minimum cost graph homomorphism problem with vertex weights defined below. Let G, H be simple graphs and $c : V(G) \times V(H) \to \mathbb{Z}$ be a homomorphism cost function associated with G. Assume that $w : V(G) \to \mathbb{N}$ is a function that assigns to every vertex $u \in V(G)$ a positive integer weight $w(v)$. For a homomorphism f of G to H, the cost of f is defined as the weighted sum below.

$$cost(f) = \sum_{v \in V(G)} w(v).c(f(v)).$$

For a fixed graph H, the *weighted minimum cost graph homomorphism problem* for H is the (optimization) problem that, given an input graph G together with a cost function

$c : V(G) \times V(H) \to \mathbb{Z}$ and positive integer vertex weights $w : V(G) \to \mathbb{N}$, asks for a homomorphism of G to H with minimum cost.

We consider the corresponding decision problem associated with the weighted minimum cost graph homomorphism problem to H, that takes as input a graph G together with a cost function c, vertex weights w, and an integer k and asks whether there is a homomorphism of G to H that costs at most k. We assume that vertex weights are bounded by a polynomial in the size of the input graph G, and show that the weighted minimum cost graph homomorphism problem (with constrained costs) and the minimum cost graph homomorphism problem (with constrained costs) are polynomially equivalent.

Theorem 5.3. *Let H be any fixed graph. Then, the weighted minimum cost graph homomorphism problem to H and MinHOM(H) are polynomially equivalent.*

Proof. We show that any instance of the weighted minimum cost graph homomorphism problem to H can be transformed, in polynomial time, into an instance of MinHOM(H). The other direction is obvious as we can simply create an instance of the weighted problem in which every vertex of the input graph has weight 1.

Let (G, c, w, k) be an instance of the weighted minimum cost graph homomorphism problem, where G is the input graph, $c : V(G) \times V(G) \to \mathbb{N}$ is the homomorphism cost function, $w : V(G) \to \mathbb{N}$ is function that gives vertex weights, and k is the maximum homomorphism cost we are interested in.

We construct a graph G' as follows. For every vertex $u \in V(G)$, replace u with a set of $w(u)$ new disjoint vertices $V_u = \{u_1, u_2, \cdots, u_{w(u)}\}$. Moreover, for every edge $e = uv$ in G, make their corresponding sets V_u, V_v in G' fully adjacent (otherwise, there will be no edges between them). Also, define $c'(u', a) = c(u, a)$ for every $a \in V(H)$, $u \in V(G)$, and $u' \in V_u$. The polynomially bounded vertex weights ensure that G' can be constructed in polynomial time, and c' can be computed in polynomial time.

For every homomorphism f of G to H, it is easy to see that there exists a homomorphism f' of G' to H such that the cost of the homomorphism f (with cost function c) is equal to the cost of homomorphism f' (with cost function c' and vertex weights $w(u)$). In fact, f' can be obtained from f by mapping all vertices corresponding to any vertex u to $f(u)$, that is, $f'(u_i) = f(u)$ for every $u_i \in V_u$.

We also show that for any homomorphism f' of G' to H, there exists a homomorphism f of G to H such that the cost of f is not more than the cost of f'. This implies that the

minimum cost homomorphism of G' to H (without vertex weights) and G to H (with vertex weights) is equal. Let $f'(V_u)$ denote the set of all images of vertices in V_u, that is, $f'(V_u) = \{f'(v)|v \in V_u\}$. Observe that when uv is an edge in G, then every $a \in f'(V_u)$ is adjacent to every vertex $b \in f'(V_v)$ $(a, b \in V(H))$. This helps us modify f' such that all vertices in V_u map to the same vertex in H without increasing the cost of homomorphism by simply choosing a vertex a in $f(V_u)$ that minimizes $c(u, a)$. We can now define a homomorphism f using this modified f' by setting $f(u) = a$ where a is the only member in $f'(V_u)$. Notice that this can also be performed in polynomial time.

□

It is possible to use a simpler construction in the proof of Theorem 5.3 that does not modify G and converts an instance of the weighted problem to an instance without weights only by changing the cost function, by defining $c'(u, a) = c(u, a) \times w(u)$. However, the transformation used in the aforementioned proof has the added benefit that preserves the constraints on the cost function. That is, whenever c is a constrained cost function (that only depends on H), c' will also be a constrained cost function (that only depends on H). Thus, we can extend Theorem 5.3 to the minimum constrained cost graph homomorphism problems as stated below.

Corollary 5.1. *Let H be any fixed graph. Then, the minimum constrained cost graph homomorphism problem to H and the weighted minimum constrained cost graph homomorphism problem to H are polynomially equivalent.*

5.3 The Hexagon

In this section, we investigate the minimum constrained cost graph homomorphism problem for graphs H that contain a cycle of length six as an induced subgraph. We present a polynomial time reduction from the *one-or-all list homomorphism problem* to H and show that the corresponding constrained problem with vertex weights is NP-complete for every such graph H.

Lemma 5.1. *Let H be a simple graph which contains a hexagon as an induced subgraph. Then, the weighted minimum constrained cost graph homomorphism problem to H is* NP*-complete.*

Recall from Section 2.2 that, for every fixed graph H, the one-or-all list homomorphism problem to H is the the list homomorphism problem to H in which every list of the input is required to be either a singleton (a list of size one) or the entire vertex set of H. In the process of studying list homomorphism problems to reflexive graphs, Feder and Hell proved in [43] that when H is a reflexive cycle of length k, with $k \geq 4$, then the one-or-all list homomorphism problem to H is NP-complete. In fact, as Feder, Hell and Huang have mentioned in their closing remarks in [46], the reduction used in the proof of Theorem 3.1 in [46] can be employed to prove a similar result for irreflexive graphs, as the only lists that they have used in their construction are either singleton lists, or the entire vertex set $V(H)$.

Lemma 5.2. *[46] Let C be the cycle of length $2k$. Then, the one-or-all list homomorphism problem to C is* NP*-complete when $k \geq 3$.*

We can now present the proof of Lemma 5.1.

Proof. The membership in NP is clear. Let $C = 1, 2, \cdots, 6$ denote the hexagon and $h_1 h_2 \cdots h_6$ be an induced subgraph of H which is isomorphic to the cycle C. We reduce from the one-or-all list homomorphism problem to C.

Let (G, L) be an instance of the one-or-all list homomorphism problem to C. That is, assume that G is a bipartite graph with $n \geq 2$ vertices and $m \geq 1$ edges, and for every vertex $v \in V(G)$, the list $L(v)$ of admissible vertices for v is either a singleton, or the set $V(C)$. We construct an instance (G', c, w, T) of the weighted minimum cost graph homomorphism problem to H and show that G has a list homomorphism to C with respect to the lists L if and only if G' has a homomorphism to H with cost less than or equal to T.

We construct the bipartite graph G' as follows. We start with a copy of G and for every vertex $v \in V(G)$ with a singleton list $L(v) = \{k\}$, we add a gadget that is the Cartesian product of v and the hexagon, using six new vertices $(v, 1), (v, 2), \cdots, (v, 6)$, and six new edges $(v, i)(v, i + 1)$ (for every $1 \leq i \leq 6$). We also connect v to exactly two vertices of its corresponding gadget by adding two more edges $v(v, k - 1)$ and $v(v, k + 1)$ (all indices modulo 6). A vertex v and its corresponding gadget is illustrated in Figure 5.1.

We define the vertex weight function w as follows.

- for every vertex v in the copy of G, let $w(v) = 1$
- for every vertex $v \in V(G)$ with a singleton list:

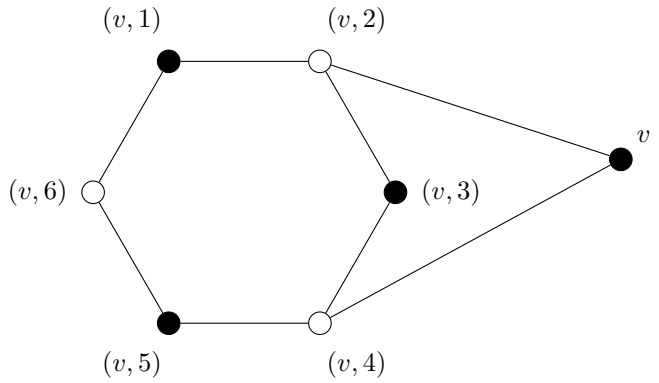

Figure 5.1: A gadget in G' for a vertex $v \in V(G)$ with a singleton list $L(v) = \{3\}$

- $w((v,1)) = w((v,4)) = 5 \times 36n^3 + 1$,
- $w((v,2)) = w((v,5)) = 1$,
- $w((v,3)) = 36n^2$,
- $w((v,6)) = 6n$.

We define the homomorphism cost function c as follows.

- $c(h_1) = c(h_4) = 0$,
- $c(h_2) = c(h_5) = 36n^2$,
- $c(h_3) = 1$,
- $c(h_6) = 6n$,
- $c(h_i) = 5 \times 36n^3 + 1$ for all other vertices $h_i \in V(H)$.

Finally, we set $T = 5 \times 36n^3 = 180n^3$ and prove that there is a list homomorphism of G to C with respect to the lists L if and only if there is a homomorphism of G' to H with cost less than or equal to T. First, assume that there is a homomorphism f of G to C with respect to the lists L. We can define a homomorphism g of G' to H as follows.

- $g(u) = h_i$ if and only if $f(u) = i$ for every vertex $u \in V(G)$ and every $1 \leq i \leq 6$, and,

- $g((u,i)) = h_i$ for every vertex $u \in V(G)$ with a singleton list $L(u) = \{k\}$ and every $1 \leq i \leq 6$.

Claim. The function g is a homomorphism of G' to H. Moreover, it only maps vertices of G' to the copy of C in H, i.e. g only uses vertices $h_1, h_2, \cdots, h_6$.

To prove the above claim, we distinguish three types of edges in G'.

1. Edges uv corresponding to the edges in G ($u, v \in V(G)$): These are clearly mapped to edges in H by g as $g(u) = f(u)$ for all vertices $u \in V(G)$ and f is a homomorphism of G to C.

2. Edges $(u,i)(u,i+1)$ that connect two vertices of the hexagon gadgets: These edges map to the corresponding edge h_ih_{i+1} by definition of g (indices modulo 6).

3. Edges that connect a vertex $u \in V(G)$ to two vertices in its corresponding gadget: Notice that there is a gadget for u in G' only when u has a singleton list $L(u) = \{i\}$. So, we have $f(u) = i$. This further implies that $g(u) = h_i$. Also, notice that $g((u,i-1)) = h_{i-1}$ and $g((u,i+1)) = h_{i+1}$ by the definition of g (again, all indices modulo 6). Hence, edges $u(u,i-1)$ and $u(u,i+1)$ also map to edges $h_{i-1}h_i$ and h_ih_{i+1}, respectively.

This completes the proof of the above Claim. We now show that the cost of g is at most $T = 180n^3$.

- For every vertex $u \in V(G)$, $w(u) = 1$ and $c(g(u)) \leq 36n^2$. Also, there are exactly n such vertices in G'. This contributes at most $36n^3$ to the cost of the homomorphism.

- For every vertex $u \in V(G)$ with a singleton list, its corresponding gadget contributes exactly $4 \times 36n^2$:

  - vertices $(u,1)$ and $(u,4)$ do not contribute to the sum, as $c(h_1) = c(h_4) = 0$,
  - vertices $(u,2)$ and $(u,5)$ each contribute $36n^2$,
  - vertices $(u,3)$ and $(u,6)$ each contributes $36n^2 = 6n \times 6n = 36n^2 \times 1$.

  There are at most n gadgets in G' (one for every vertex $u \in V(G)$), and so, the total contribution of all vertices of the gadgets is at most $4 \times 36n^3$.

Therefore, the total cost of the homomorphism g is at most $5 \times 36n^3 = 180n^3 = T$.

Conversely, let g be a homomorphism of G' to H which costs at most $T = 180n^3$. We prove that there is a homomorphism f of G to C with respect to the lists L. First, we show that g has the following two properties.

- It only maps vertices of G' to the vertices of the hexagon $h_1, h_2, \cdots, h_6$, and,
- every gadget in G' is mapped identically to the hexagon in H, that is, for every vertex $u \in V(G)$ with a singleton list $L(u)$, and for every $1 \leq i \leq 6$, $g((u,i)) = h_i$.

The first property holds because $c(a) > T$ for every vertex $a \in V(H)$ other than the vertices of the hexagon (and the fact that, by definition, all vertex weights are positive integers). In fact, we must have $w(u) \times c(g(u)) \leq T$, or equivalently, $c(g(u)) < \frac{(T+1)}{w(u)}$, for every vertex $u \in V(G')$. This restricts possible images of vertices with large vertex weights. Consider vertices in the gadget of a vertex $u \in V(G')$. For instance, every $(u,4)$ must map to either h_1 or h_4. Similarly, none of the $(u,3)$ vertices can map to any vertex other than h_1, h_3, or h_4. Given that $(u,3)$ and $(u,4)$ are adjacent in G', their images must also be adjacent in H. This enforces $f((u,3)) = h_3$ and $f((u,4)) = h_4$ (for every u that has a gadget in G'). Similar to $(u,4)$, g must also map every $(u,1)$ to either h_1 or h_4, but $g((u,1)) = h_4$ is not feasible as it does not leave any options for the image of $(u,2)$. Hence, $g((u,1)) = h_1$. This further implies that $g((u,6)) = h_6$ (as it is adjacent to $(u,1)$), and finally, $g((u,2)) = h_2$ and $g((u,5)) = h_5$.

It is now easy to verify that for every vertex $u \in V(G)$ with a singleton list $L(u) = j$, we always have $g(u) = h_j$. This is because u is adjacent to $(u, j-1)$ and $(u, j+1)$ in G' and the only vertex in H that is adjacent to the $g((u,j-1)) = h_{j-1}$ and $g((u,j+1)) = h_{j+1}$ and cost of mapping to it is less than or equal to T is h_j. This completes the proof as we can define a homomorphism f of G to C with respect to the lists L as $f(v) = i \iff g(v) = h_i$. □

A shorthand of the construction used in the proof of Lemma 5.1 is shown in Figure 5.2.

5.4 Large Even Cycles

In this section, we extend our results in Section 5.3 to graphs H that contain even cycles of length at least 8. We use a reduction similar to the one discussed in the previous section to prove the following Lemma.

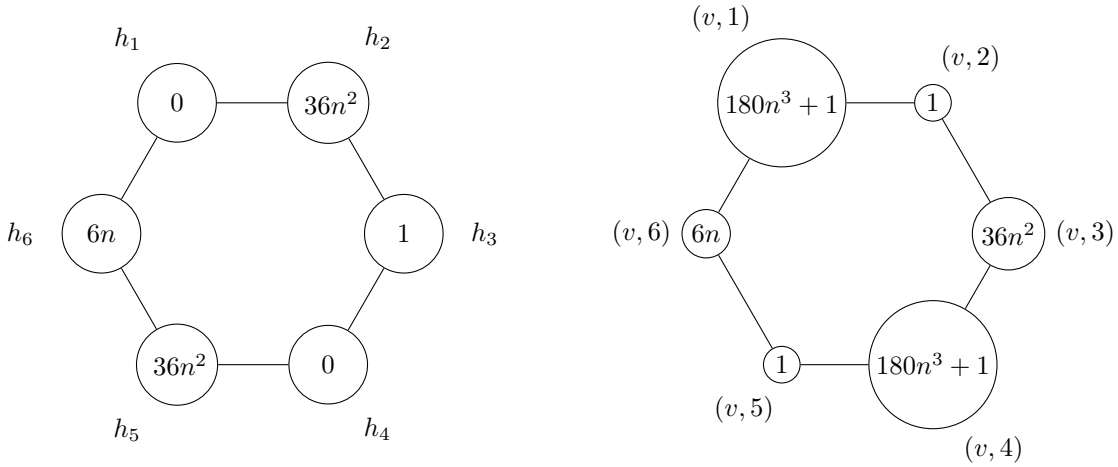

Figure 5.2: A hexagon in H together with associated homomorphism costs (left), and a gadget in G' together with vertex weights (right).

Lemma 5.3. *Let H be a simple bipartite graph which contains a cycle of length at least eight as an induced subgraph. Then, the weighted minimum constrained cost graph homomorphism problem to H is* NP*-complete.*

Proof. Again, the membership in NP is clear. Let $C = 1, 2, \cdots, 2k$ be an even cycle, and $h_1 h_2 \cdots h_{2k}$ be an induced subgraph of H which is isomorphic to C ($k \geq 4$). Again, we reduce from the all-or-one list homomorphism problem to C.

We take an instance of the one-or-all list homomorphism problem to C, i.e., a bipartite graph G with $n \geq 2$ vertices and $m \geq 1$ edges, and lists $L(v)$ that are either a singleton, or the entire set $V(C)$. We construct a corresponding instance (G', c, w, T) of the weighted minimum cost graph homomorphism problem to H.

The bipartite graph G' is constructed exactly as before: start with a copy of G and for every vertex v with a list $L(v) = \{t\}$, we add the Cartesian product of v and C using $2k$ new vertices and $2k$ new edges. Finally, we make v adjacent to two vertices in its corresponding gadget, $(v, t-1)$ and $(v, t+1)$ (all indices modulo $2k$). An example of a vertex with a singleton list and its gadget in G' is shown in Figure 5.3.

We define the vertex weight function w as follows. An example of a gadget with vertex weights is illustrated in Figure 5.4.

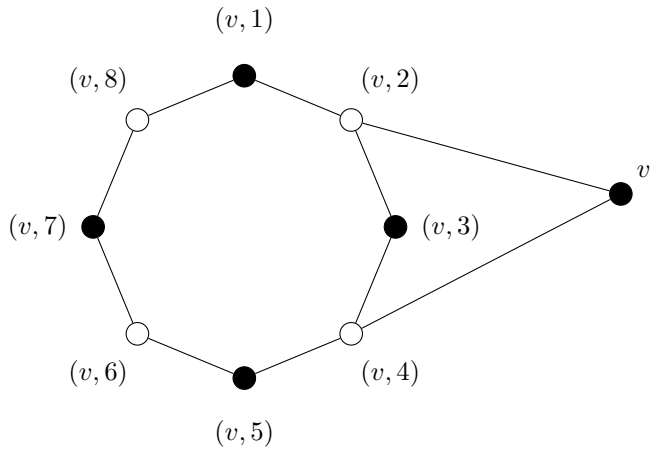

Figure 5.3: A gadget in G' corresponding to a vertex v with a singleton list $L(v) = \{c_3\}$ $(k = 4)$

- for every vertex v in the copy of G, let $w(v) = 1$
- for every vertex $v \in V(G)$ with a singleton list:
  - $w((v,1)) = w((v,4)) = 50kn^2$,
  - $w((v,2)) = w((v,3)) = w((v,5)) = 1$,
  - $w((v,i)) = 9n$ for all $6 \leq i \leq 2k$

We define the homomorphism cost function c as follows (See Figure 5.5).

- $c(h_1) = c(h_4) = 0$,
- $c(h_2) = c(h_3) = c(h_5) = 8kn$,
- $c(h_i) = 1$ for all $6 \leq i \leq 2k$,
- $c(h_i) = 50kn^2$ otherwise.

Finally, we set $T = 50kn^2 - 1$ and argue that there is a list homomorphism of G to C with respect to the lists L if and only if there is a homomorphism of G' to H with cost less than or equal to T. The rest of the proof is similar to the proof of Lemma 5.1.

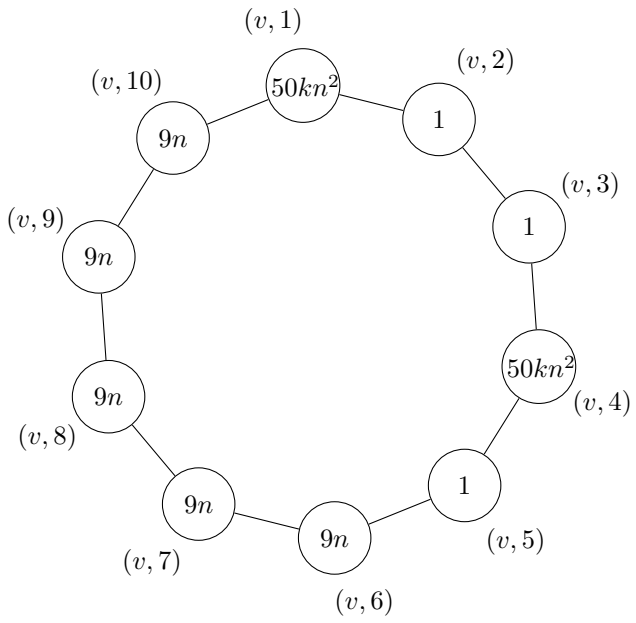

Figure 5.4: Vertex weights for a gadget in G' ($k = 5$)

First, assume that there is a homomorphism f of G to C with respect to the lists L. A homomorphism $g : V(G') \to V(H)$ can easily be constructed by mapping each vertex $u \in V(G)$ to h_i whenever $f(u) = i$, and mapping each gadget in G' identically to the copy of C in H (i.e., $g((u,i)) = h_i$). We now show that the cost of g does not exceed T.

- For every vertex $u \in V(G)$, $w(u) = 1$ and $c(g(u)) \leq 8kn$.
- For every vertex $u \in V(G)$ with a singleton list, its corresponding gadget contributes exactly $42kn - 45n = 3 \times 8kn + (2k-5) \times 9n$:
  - vertices $(u,1)$ and $(u,4)$ do not contribute to the sum,
  - vertices $(u,2)$, $(u,3)$ and $(u,5)$ each contribute $8kn$,
  - vertices (u,i) each contribute $9n$ ($6 \leq i \leq 2k$).

This gives an upper bound of $n \times (50kn - 45n) < 50kn^2$ on the cost of g (given that G has at least one vertex).

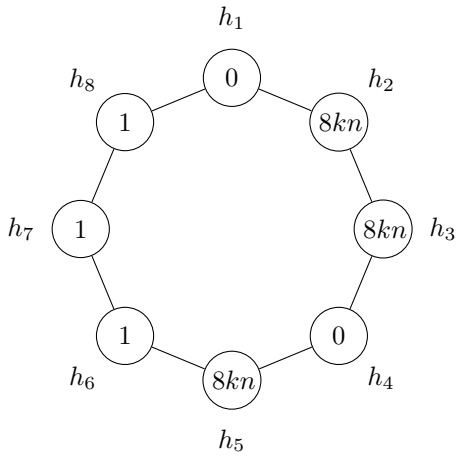

Figure 5.5: An even cycle in H with costs ($k = 4$)

Conversely, let g be a homomorphism of G' to H which costs less than $50kn^2$. Observe that g can only map vertices of G' to the copy of C in H. This is because every vertex weight is at least 1 and the cost of all other vertices in H is $50kn^2$.

We show that every gadget maps identically to the copy of C in H. Again, we must have $c(g(u)) < \frac{(T+1)}{w(u)}$ for every vertex $u \in V(G')$. Hence, $(u, 1)$ and $(u, 4)$ must map to h_1 and h_4, and they cannot map to the same vertex because they belong to different partitions in G'. Also, for every u and every $i > 5$, the vertex (u, i) cannot map to h_2, h_3, or h_5. Specifically, $(u, 2k)$ which is adjacent to $(u, 1)$ can only map to a neighbour of h_1 or h_4 other than h_2, h_3, and h_5. This enforces $g((u, 2k)) = h_{2k}$, which implies $g((u, 1)) = h_1$ and $g((u, 4)) = h_4$. This further implies $g((u, 2)) = h_2$ and $g((u, 3)) = h_3$ (because the other path between h_1 and h_4 has length $2k - 3 \geq 5$). It is now clear that the for all remaining vertices (u, i) (with $i > 5$), we have $g((u, i)) = h_i$.

Now, we can conclude that $g(u) = h_j$ for every vertex u with a singleton list $L(u) = \{c_j\}$, because u is adjacent to $(u, j - 1)$ and $(u, j + 1)$ in G'. This completes the proof as we can define a homomorphism f of G to C with respect to the lists L as $f(v) = i$ whenever $g(v) = h_i$ for every $v \in V(G)$. □

Note that the proof of Lemma 5.3 does not apply when $k = 3$. In fact, the corresponding minimum cost homomorphism problem is polynomial time solvable for the inputs

constructed in the reduction. This is because the two paths between c_1 and c_4 have the same length, and hence, no homomorphism of minimum cost will use c_2 (thus, the problem reduces to the minimum cost graph homomorphism to a path of length 6, which is known to be efficiently solvable).

Theorem 5.4. *Let H be a bipartite graph.*
If H is a proper interval bigraph, then the minimum constrained cost graph homomorphism problem to H is in P.
If H is not chordal bipartite, then the minimum constrained cost graph homomorphism problem to H is NP*-complete.*

Proof. If every component of H is a proper interval bigraph, then problem without constrained costs, MinHOM(H), is polynomial time solvable [62] (Theorem 2.7). If H is not chordal bipartite, then, by definition, it contains an even cycle of length at least six as an induced subgraph. Hence, this is an immediate result of Corollary 5.1 and Lemmata 5.1 and 5.3. □

We note that Theorem 5.4 gives only a partial dichotomy for the minimum constrained cost graph homomorphism problem, as there is a gap between the class of chordal bipartite graphs and the class of proper interval bigraphs. Specifically, the class of proper interval bigraphs consists precisely of those chordal bipartite graphs that do not contain a bipartite claw, a bipartite net, or a bipartite tent as an induced subgraph (see Lemma 2.1). In the next Section, we make this gap smaller, and in particular, we close the gaps for trees.

5.5 The Dichotomy For Trees

In this section, we extend Theorem 5.4 to graphs H that contain a bipartite claw. As in the case of large cycles, we focus on the weighted version of the problem and show that it is NP-complete when the target graph H contains a bipartite claw. This helps us determine a dichotomy classification for trees.

Lemma 5.4. *Let H be a fixed simple graph which contains the bipartite claw as an induced subgraph. The weighted minimum constrained cost graph homomorphism problem to H is* NP*-complete.*

It is well known that, given as input a graph G and an integer k, the problem of deciding whether G has an independent of size k is NP-complete. Gutin, Hell, Rafiey and Yeo [62] proved that the problem is still NP-complete even when the input is restricted to be a 3-partite graph.

Theorem 5.5. *[62] The problem of finding a maximum independent set in a 3-partite graph G, even given the three partite sets, in* NP*-complete.*

The main idea of the proof of Lemma 5.4 is similar to the proofs of Lemmata 5.1 and 5.3. We show that finding an independent set of size at least k in an arbitrary 3-partite graph G is equivalent to finding a homomorphism of cost at most k' in an auxiliary graph G' together with constrained costs c and vertex weights w. To construct G', we start by adding a fixed number of placeholder vertices; vertices that, with the appropriate weights and costs, always map to the same specific vertices of the target graph H in any homomorphism of G' to H of minimum cost. We then use these placeholder vertices in our construction to ensure that the vertices corresponding to each part of the the input graph G are only mapped to certain vertices of the target graph H.

We can now present the proof of Lemma 5.4.

Proof. The membership in NP is clear. To show that the problem is NP-hard, we reduce from the problem of finding a maximum independent set in a 3-partite graph, stated in Theorem 5.5. Let G be a 3-partite graph in which we seek an independent set of size k, with parts V_1, V_2, and V_3, and denote by and n and m the number of vertices and edges in G, respectively. We assume that G is non-empty. Without loss of generality, we can assume that $|V_1| \geq 1$. We construct an instance $(G', c, w, T_{G,k})$ of the weighted minimum cost graph homomorphism and show that G has an independent set of size k if and only if there is a homomorphism of G' to H with cost less than or equal to $T_{G,k}$.

We construct the bipartite graph G' as follows. Subdivide every edge e in G using a new vertex d_e (which is adjacent to both ends of e). Add three vertices b_1, b_2 and b_3 and make each b_i adjacent to all vertices in V_i for $i = 1, 2, 3$. Finally, add three more vertices c_0, c_1 and c_2. Make c_0 adjacent to b_1, b_2 and b_3, c_1 adjacent to b_1 and c_2 adjacent to b_2. A 3-partite graph G together with its corresponding G' is depicted in Figure 5.6. For future reference, we denote the set $\{b_1, b_2, b_3, c_0, c_1, c_2\}$ by V_4.

Let $H' = (X, Y)$ be an induced subgraph of H which is isomorphic to a bipartite claw

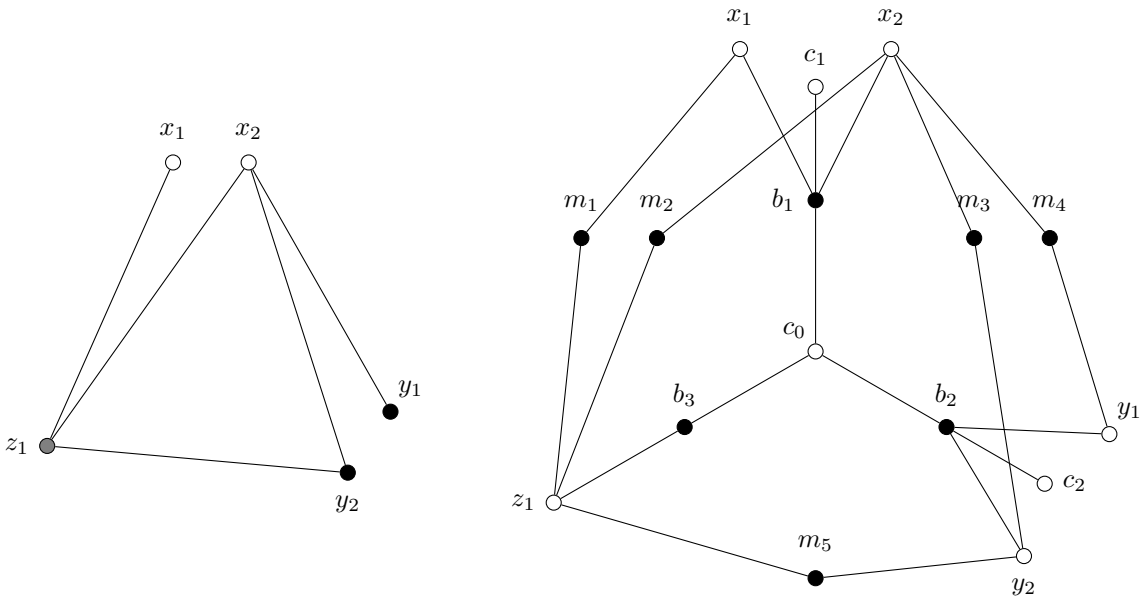

Figure 5.6: A 3-partite graph G with parts $V_1 = \{x_1, x_2\}$, $V_2 = \{y_1, y_2\}$, $V_3 = \{z_1\}$ (left) and its corresponding bipartite graph G' (right)

with parts $X = \{v_0, v_1, v_2, v_3\}$ and $Y = \{u_1, u_2, u_3\}$, and edge set

$$E' = \{u_1v_1, u_2v_2, u_3v_3, u_1v_0, u_2v_0, u_3v_0\}.$$

Define the homomorphism cost function c as follows.

- $c(v_0) = 4$
- $c(v_1) = c(u_1) = 1$
- $c(u_2) = c(v_3) = 3$
- $c(v_2) = c(u_3) = 0$
- $c(u) = 160n(m + n)$ for every other vertex $u \notin X \cup Y$

This is illustrated in Figure 5.7.

Define the vertex weights of G' as follows.

- $w(b_1) = w(c_1) = 50n(m + n)$

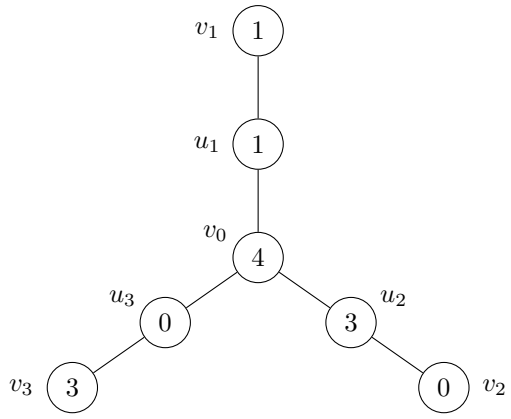

Figure 5.7: A bipartite claw, with homomorphism costs

- $w(b_3) = w(c_2) = 160n(m+n)$
- $w(b_2) = w(c_0) = 1$
- $w(u) = 4(m+n)$ for every vertex $u \in V_1$
- $w(u) = 3(m+n)$ for every vertex $u \in V_2$
- $w(u) = 12(m+n)$ for every vertex $u \in V_3$

Finally, let $T_{G,k}$ be the sum of the following values.

- $T^1_{G,k} = 16(m+n)|V_1|$,
- $T^2_{G,k} = 12(m+n)|V_2|$,
- $T^3_{G,k} = 48(m+n)|V_3|$,
- $T^4_{G,k} = 2 \times 50n(m+n) + 4 + 3$,
- $T^e_{G,k} = 3m$, and,
- $T^I_{G,k} = -12(m+n)k$.

Or equivalently:

$$T_{G,k} = 100n(m+n) + 7 + 3m + (4|V_1| + 36|V_3|)(m+n) + 12(m+n)(n-k)$$

We prove that G has an independent set of size k if and only if there is a homomorphism of G' to H of cost less than or equal to $T_{G,k}$.

First, assume that I is an independent set of size k in G with parts $I_1 \subset V_1$, $I_2 \subset V_2$, and $I_3 \subset V_3$. Let k_i denote $|I_i|$ $(i = 1, 2, 3)$. Define the homomorphism f_I as follows.

- $f_I(u) = v_i$ for all vertices $u \in I_i$ $(i = 1, 2, 3)$,
- $f_I(u) = v_0$ for all vertices $u \in V(G) - I$,
- $f_I(d_e) = u_j$ for every edge e with one end in I_j $(j = 1, 2, 3)$,
- $f_I(d_e) = u_3$ for every edge e with both ends in $V - I$,
- $f_I(b_j) = u_j$ for $j = 1, 2, 3$, and finally,
- $f_I(c_k) = v_k$ for $k = 0, 1, 2$.

Notice that at most one end of each edge e is in the independent set, hence, the above assignment is indeed a function. In fact, it is easy to verify that f_I is a homomorphism.

- edges subdivided from edges e with both ends in $V - I$ map to the edge v_0u_3,
- edges subdivided from edges e with one end in I_i and the other end in $V - I$ map to u_iv_i and u_iv_0 $(i = 1, 2, 3)$,
- edges connecting b_i to V_i map to u_iv_i $(i = 1, 2, 3)$,
- c_0b_i map to v_0u_i $(i = 1, 2, 3)$, and,
- b_ic_i map to v_iu_i $(i = 1, 2)$.

We now compute the cost of the homomorphism f_I and show that it is at most $T_{G,k}$.

- The vertices in V_1 contribute exactly $(|V_1| - k_1) \times 16(m + n) + k_1 \times 4(m + n)$, or, $T^1_{G,k} - 12k_1(m + n)$,
- the vertices in V_2 contribute exactly $(|V_2| - k_2) \times 12(m + n) + k_1 \times 0$, or, $T^2_{G,k} - 12k_2(m + n)$,
- the vertices in V_3 contribute exactly $(|V_3| - k_3) \times 48(m + n) + k_3 \times 36(m + n)$, or, $T^3_{G,k} - 12k_3(m + n)$,

- the vertices in V_4 contribute a total of $100n(m+n)+7 = T^4_{G,k}$ (see Table 5.1), and,
- the vertices d_e contribute at most $3m = T^e_{G,k}$.

Notice that $k = k_1 + k_2 + k_3$, hence, the cost of the homomorphism f_I is at most $T_{G,k}$.

vertex v	$w(v)$	$f_I(v)$	$c(f_I(v))$	contributed cost of v
b_1	$50n(m+n)$	u_1	1	$50n(m+n)$
b_2	1	u_2	3	3
b_3	$160n(m+n)$	u_3	0	0
c_0	1	v_0	4	4
c_1	$50n(m+n)$	v_1	1	$50n(m+n)$
c_2	$160n(m+n)$	v_2	0	0

Table 5.1: contribution of vertices in V_4 to the cost of homomorphism f_I

Conversely, assume that f is a homomorphism of G' to H which costs less than or equal to $T_{G,k}$. Note that $T_{G,k} < 150n(m+n)$. This prevents any vertex v to map to a vertex a when $c(v,a) \times w(v) \geq T_{G,k}$. In particular, b_1 and c_1 can only map to vertices a with $c(a) < 3$, i.e, v_1, u_1, v_2, u_3. But b_1 and c_1 are adjacent and the only edge in H among these four vertices is u_1v_1. Similarly, b_3 and c_2 can only map to u_3 or v_2. Observe that $f(b_3) = v_2$ is not feasible, as it implies $f(c_0) = u_2$ and hence $f(b_1) \in \{v_0, v_2\}$. Thus, we have $f(b_3) = u_3$, $f(b_1) = u_1$, $f(c_1) = v_1$, $f(c_0) = v_0$, $f(c_2) = v_2$, and finally $f(b_2) = u_2$.

This restricts possible images of vertices in V. Specifically, all vertices in V_1 are adjacent to b_1, thus, f can only map them to v_1 or v_0, the neighbourhood of $u_1 = f(b_1)$. Similarly, each vertex in V_2 will only map to v_2 or v_0, and each vertex in V_3 will only map to v_3 or v_0.

Let I denote the set of vertices of G that f maps to v_1, v_2 or v_3. Notice that I is an independent set in G. This is because any two adjacent vertices in G are of distance two in G' but the shortest path between v_1 and v_2, or between v_2 and v_3, or between v_3 and v_1 in H' has length 4.

We complete the proof by showing that $|I| \geq k$. Let $|I| = k'$ and assume for a contradiction that $k' < k$. Let f_I denote the homomorphism of G' to H constructed from I as described in the first part of the proof with $cost(f_I) \leq T_{G,k'}$. Observe that f and f_I are identical for every vertex $v \in V_i$ $(i = 1, 2, 3, 4)$. Hence, $|cost(f) - cost(f_I)| \leq 3m$. This implies that $cost(f_I) \leq cost(f) + 3m$. Also, note that $cost(f_I) \geq T_{G,k'} - 3m$, hence, we have $T_{G,k'} - 3m \leq T_{G,k} + 3m$, or equivalently, $T_{G,k'} - T_{G,k} \leq 6m$. But this is a contradiction

because:

$$T_{G,k'} - T_{G,k} = T^I_{G,k'} - T^I_{G,k} = 12(m+n)(k-k') \geq 12(m+n).$$

□

We can now apply Corollary 5.1 and conclude the following Theorem for the problem without vertex weights.

Theorem 5.6. *Let H be a fixed simple graph which contains the bipartite claw as an induced subgraph. Then, the minimum constrained cost graph homomorphism problem to H is* NP*-complete.*

Interestingly, the only forbidden structure of proper interval bigraphs (discussed in Lemma 2.1) that does not contain a cycle, is the bipartite claw. Thus, we are now able to completely distinguish between easy and hard minimum cost graph homomorphism problems to trees.

Theorem 5.7. *Let T be a fixed tree. The minimum constrained cost graph homomorphism problem to T is in* P *when T is a proper interval bigraph, and is* NP*-complete otherwise.*

Proof. If T is a proper interval bigraph, then the problem without constrained costs is also polynomial (Theorem 2.7). If T is not a proper interval bigraph, then it must contain a bipartite claw. Thus, the problem is NP-complete by Theorem 5.6. □

Chapter 6

Conclusion and Future Work

6.1 Counting Matrix Partitions

We leave open the counting complexity of matrix partition problems in general. In particular, we do not know whether a dichotomy classification exists for matrices M of size greater than four. Recall from Chapter 2 that Dyer, Goldberg and Richerby have recently extended our results for small matrices to matrices of size 4 by means of a computer-assisted proof [37].

Our dichotomy classification for counting M-partitions for small matrices coincides with the dichotomy recently discovered by Gobel, Goldberg, McQuillan, Richerby and Yamakami for the list version of the problems (discussed in Section 3.5). This suggests another open question, as conjectured by Dyer, Goldberg and Richerby in [37]. Are these two problems, counting M-partitions, and counting list M-partitions, polynomially equivalent for every matrix M? It is known that the answer to the above question is true for matrices M that represent graph homomorphism problems [76], and also for matrices M of size at most 4 (Theorems 3.9 and 3.13).

Another interesting open area to investigate is the question of monotonicity of M-partition problems. Is M-partition $\#$ P-complete when it contains a principal submatrix M' such that M'-partition is $\#$ P-complete? In the case of list matrix partition problems, monotonicity is straight forward because we can always avoid putting vertices in some parts using appropriate lists. But this is not necessarily true when we do not have lists.

6.2 Approximation Algorithms

In Chapter 4, we have classified the complexity of approximately solving minimum cost graph homomorphism problems for general graphs (that contain vertices with loops and vertices without loops), as well as for directed graphs, but the approximation ratio guaranteed by our algorithm is n, the size of the input graph. It would be interesting to improve the approximation ratio in both of these cases. In particular, we do not know if there exists any constant ratio approximation algorithm for these problems when the target graph H is a directed graph or a general undirected graph (that contains both a vertex with loop and a vertex without loop), and hence, whether they belong to the complexity class APX.

In the case of co-circular arc bigraphs and interval graphs where we have presented specific constant ratio approximation algorithms, the most interesting open question is whether the approximation ratio can be bounded by a constant independent of H. Our algorithm is both a $|V(H)|$-approximation and a $1 + |E'|$-approximation algorithm. These are constants independent of the input (the graph G and the homomorphism cost function c), but very much dependent on the fixed graph H. For many bipartite graphs H (including the bipartite tent, net, or claw), one can choose a starting min ordering such that $|E'| = 1$, thus obtaining a 2-approximation algorithm. With a bit more effort it can be shown that a 2-approximation algorithm exists for the so-called doubly convex bipartite graphs. We have not excluded the possibility that there exist polynomial time 2-approximation algorithms (or k-approximation algorithms, for some absolute constant k) for all co-circular arc bipartite graphs or all interval graphs. Until such a possibility is excluded, there is not much interest in making slight improvements to the approximation ratio.

Finally, we leave open the complexity of approximately solving the bottleneck minimum cost graph homomorphism problem. Trivially, unless P = NP, there is no efficient approximation with a multiplicative guarantee possible for this problem when the graph homomorphism problem is NP-complete, as one can use an identity cost function $c = 1$ and solve HOM(H) otherwise.

Theorem 6.1. *Let H be a fixed graph. When H is not bipartite, then there is no polynomial time approximation algorithm with multiplicative guarantee possible for BMinHOM(H), unless* P = NP.

6.3 Minimum Constrained Cost Graph Homomorphisms

We leave open the complexity of the minimum constrained cost graph homomorphism problems in general. In particular, we do not know the complexity of the problem when H contains a bipartite net, or a bipartite tent (but does not contain a bipartite claw, or an even cycle of length at least six). These problems are specifically interesting because these are the only remaining forbidden structures of proper interval bigraphs (Lemma 2.1). In fact, it will be surprising if both of these problems are NP-complete, as we have the following result.

Theorem 6.2. *Suppose that the minimum constrained cost graph homomorphism problems to bipartite net and to bipartite tent are both* NP*-complete. Then, for every bipartite graph H, the minimum constrained cost graph homomorphism problem H and MinHOM(H) are polynomially equivalent.*

Bibliography

[1] Gagan Aggarwal, Tomás Feder, Rajeev Motwani, and An Zhu. Channel assignment in wireless networks and classification of minimum graph homomorphism. *ECCC TR06*, 40, 2006.

[2] Michael O. Albertson and Joan P. Hutchinson. Extending colorings of locally planar graphs. *Journal of Graph Theory*, 36(2):105–116, 2001.

[3] Michael O. Albertson and Emily H. Moore. Extending graph colorings. *J. Comb. Theory, Ser. B*, 77(1):83–95, 1999.

[4] Vladimir E. Alekseev and Vadim V. Lozin. Independent sets of maximum weight in (p, q)-colorable graphs. *Discrete Mathematics*, 265(1-3):351–356, 2003.

[5] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[6] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and hardness of approximation problems. In *FOCS*, pages 14–23, 1992.

[7] Hans-Jürgen Bandelt, Martin Farber, and Pavol Hell. Absolute reflexive retracts and absolute bipartite retracts. *Discrete Applied Mathematics*, 44(1-3):9–20, 1993.

[8] H.J Bandelt, A Dhlmann, and H Schtte. Absolute retracts of bipartite graphs. *Discrete Applied Mathematics*, 16(3):191 – 215, 1987.

[9] Jørgen Bang-Jensen and Pavol Hell. On chordal proper circular arc graphs. *Discrete Mathematics*, 128(1-3):395–398, 1994.

[10] Amotz Bar-Noy, Mihir Bellare, Magnús M. Halldórsson, Hadas Shachnai, and Tami Tamir. On chromatic sums and distributed resource allocation. *Inf. Comput.*, 140(2):183–202, 1998.

[11] Amotz Bar-Noy and Guy Kortsarz. Minimum color sum of bipartite graphs. *J. Algorithms*, 28(2):339–365, 1998.

[12] M. Biró, Mihály Hujter, and Zsolt Tuza. Precoloring extension. i. interval graphs. *Discrete Mathematics*, 100(1-3):267–279, 1992.

[13] Manuel Bodirsky, Jan Kára, and Barnaby Martin. The complexity of surjective homomorphism problems - a survey. *Discrete Applied Mathematics*, 160(12):1680–1690, 2012.

[14] Adrian Bondy and U.S.R. Murty. *Graph Theory (Graduate Texts in Mathematics)*. Springer, 2008.

[15] Flavia Bonomo, Guillermo Durán, and Javier Marenco. Exploring the complexity boundary between coloring and list-coloring. *Annals OR*, 169(1):3–16, 2009.

[16] Kellogg S. Booth and George S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. *J. Comput. Syst. Sci.*, 13(3):335–379, 1976.

[17] Christian Borgs, Jennifer Chayes, Lszl Lovsz, VeraT. Ss, and Katalin Vesztergombi. Counting graph homomorphisms. In Martin Klazar, Jan Kratochvl, Martin Loebl, Ji Matouek, Pavel Valtr, and Robin Thomas, editors, *Topics in Discrete Mathematics*, volume 26 of *Algorithms and Combinatorics*, pages 315–371. Springer Berlin Heidelberg, 2006.

[18] Richard C. Brewster and Gary MacGillivray. Building blocks for the variety of absolute retracts. *Discrete Mathematics*, 306(15):1758–1764, 2006.

[19] Andrei Bulatov, Peter Jeavons, and Andrei Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM J. Comput.*, 34(3):720–742, March 2005.

[20] Andrei A. Bulatov. Tractable conservative constraint satisfaction problems. In *LICS*, pages 321–330, 2003.

[21] Andrei A. Bulatov. H-coloring dichotomy revisited. *Theor. Comput. Sci.*, 349(1):31–39, 2005.

[22] Andrei A. Bulatov. The complexity of the counting constraint satisfaction problem. *J. ACM*, 60(5):34:1–34:41, October 2013.

[23] Andrei A. Bulatov and Víctor Dalmau. Towards a dichotomy theorem for the counting constraint satisfaction problem. *Inf. Comput.*, 205(5):651–678, May 2007.

[24] Kathie Cameron, Elaine M. Eschen, Chính T. Hoàng, and R. Sritharan. The complexity of the list partition problem for graphs. *SIAM J. Discrete Math.*, 21(4):900–929, 2007.

[25] Pierre Charbit, Fabien de Montgolfier, and Mathieu Raffinot. Linear time split decomposition revisited. *SIAM J. Discrete Math.*, 26(2):499–514, 2012.

[26] Hubie Chen. An algebraic hardness criterion for surjective constraint satisfaction. *Algebra universalis*, 72(4):393–401, 2014.

[27] Vasek Chvátal. Star-cutsets and perfect graphs. *J. Comb. Theory, Ser. B*, 39(3):189–199, 1985.

[28] D.G. Corneil and P.A. Kamula. Extensions of permutation and interval graphs. Combinatorics, graph theory, and computing, Proc. 18th Southeast. Conf., Boca Raton/Fl. 1987, Congr. Numerantium 58, 267-275 (1987)., 1987.

[29] Alain Cournier and Michel Habib. *A new linear algorithm for Modular Decomposition.* Springer-Verlag, 1994.

[30] Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. A polynomial algorithm for 3-compatible coloring and the stubborn list partition problem (the stubborn problem is stubborn no more). *SIAM J. Comput.*, 41(4):815–828, 2012.

[31] Celina M. Herrera de Figueiredo. The p versus np-complete dichotomy of some challenging problems in graph theory. *Discrete Applied Mathematics*, 160(18):2681–2693, 2012.

[32] Celina M. Herrera de Figueiredo, Sulamita Klein, Yoshiharu Kohayakawa, and Bruce A. Reed. Finding skew partitions efficiently. *J. Algorithms*, 37(2):505–521, 2000.

[33] Celina M. Herrera de Figueiredo, João Meidanis, and Célia Picinin de Mello. A linear-time algorithm for proper interval graph recognition. *Inf. Process. Lett.*, 56(3):179–184, 1995.

[34] Xiaotie Deng, Pavol Hell, and Jing Huang. Linear-time representation algorithms for proper circular-arc graphs and proper interval graphs. *SIAM J. Comput.*, 25(2):390–403, 1996.

[35] Guillermo Duran, Luciano N. Grippo, and Martin D. Safe. Structural results on circular-arc graphs and circle graphs: A survey and the main open problems. *Discrete Applied Mathematics*, (0):–, 2013.

[36] Martin E. Dyer, Leslie Ann Goldberg, and Mike Paterson. On counting homomorphisms to directed acyclic graphs. *J. ACM*, 54(6), 2007.

[37] Martin E. Dyer, Leslie Ann Goldberg, and David Richerby. Counting 4×4 matrix partitions of graphs. *CoRR*, abs/1407.7799, 2014.

[38] Martin E. Dyer and Catherine S. Greenhill. The complexity of counting graph homomorphisms. *Random Struct. Algorithms*, 17(3-4):260–289, 2000.

[39] Martin E. Dyer and Catherine S. Greenhill. Corrigendum: The complexity of counting graph homomorphisms. *Random Struct. Algorithms*, 25(3):346–352, 2004.

[40] Martin R. Ehmsen and Kim S. Larsen. A technique for exact computation of precoloring extension on interval graphs. *Int. J. Found. Comput. Sci.*, 24(1):109–122, 2013.

[41] P. Erdős, A. L. Rubin, and H. Taylor. Choosability in graphs. *Congressus Numerantium*, 26:125–157, 1979.

[42] Tomás Feder. Classification of homomorphisms to oriented cycles and of k-partite satisfiability. *SIAM J. Discrete Math.*, 14(4):471–480, 2001.

[43] Tomás Feder and Pavol Hell. List homomorphisms to reflexive graphs. *J. Comb. Theory, Ser. B*, 72(2):236–250, 1998.

[44] Tomás Feder and Pavol Hell. Matrix partitions of perfect graphs. *Discrete Mathematics*, 306(19-20):2450–2460, 2006.

[45] Tomás Feder, Pavol Hell, and Winfried Hochstattler. Generalized colourings (matrix partitions) of cographs. In Adrian Bondy, Jean Fonlupt, Jean-Luc Fouquet, Jean-Claude Fournier, and JorgeL. Ramrez Alfonsn, editors, *Graph Theory in Paris*, Trends in Mathematics, pages 149–167. Birkhuser Basel, 2007.

[46] Tomás Feder, Pavol Hell, and Jing Huang. List homomorphisms and circular arc graphs. *Combinatorica*, 19(4):487–505, 1999.

[47] Tomás Feder, Pavol Hell, and Jing Huang. Bi-arc graphs and the complexity of list homomorphisms. *Journal of Graph Theory*, 42(1):61–80, 2003.

[48] Tomás Feder, Pavol Hell, Jing Huang, and Arash Rafiey. Interval graphs, adjusted interval digraphs, and reflexive list homomorphisms. *Discrete Applied Mathematics*, 160(6):697–707, 2012.

[49] Tomás Feder, Pavol Hell, Peter Jonsson, Andrei A. Krokhin, and Gustav Nordh. Retractions to pseudoforests. *SIAM J. Discrete Math.*, 24(1):101–112, 2010.

[50] Tomás Feder, Pavol Hell, Sulamita Klein, and Rajeev Motwani. List partitions. *SIAM J. Discrete Math.*, 16(3):449–478, 2003.

[51] Tomás Feder, Pavol Hell, Sulamita Klein, Loana Tito Nogueira, and Fábio Protti. List matrix partitions of chordal graphs. *Theor. Comput. Sci.*, 349(1):52–66, 2005.

[52] Tomás Feder, Pavol Hell, and Kim Tucker-Nally. Digraph matrix partitions and trigraph homomorphisms. *Discrete Applied Mathematics*, 154(17):2458–2469, 2006.

[53] Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic snp and constraint satisfaction: A study through datalog and group theory. *SIAM J. Comput.*, 28(1):57–104, 1998.

[54] Mathew C. Francis, Pavol Hell, and Juraj Stacho. Forbidden structure characterization of circular-arc graphs and a certifying recognition algorithm. *CoRR*, abs/1408.2639, 2014.

[55] Tibor Gallai. Transitiv orientierbare graphen. *Acta Mathematica Hungarica*, 18(1):25–66, 1967.

[56] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

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

[58] Andreas Gobel, Leslie Ann Goldberg, Colin McQuillan, David Richerby, and Tomoyuki Yamakami. Counting list matrix partitions of graphs. *2014 IEEE 29th Conference on Computational Complexity (CCC)*, Jun 2014.

[59] Petr A. Golovach, Daniel Paulusma, and Jian Song. Computing vertex-surjective homomorphisms to partially reflexive trees. *Theor. Comput. Sci.*, 457:86–100, 2012.

[60] M.C. Golumbic. *Algorithmic graph theory and perfect graphs.* Computer science and applied mathematics. Academic Press, 1980.

[61] Arvind Gupta, Pavol Hell, Mehdi Karimi, and Arash Rafiey. Minimum cost homomorphisms to reflexive digraphs. In *LATIN*, pages 182–193, 2008.

[62] Gregory Gutin, Pavol Hell, Arash Rafiey, and Anders Yeo. A dichotomy for minimum cost graph homomorphisms. *Eur. J. Comb.*, 29(4):900–911, 2008.

[63] Gregory Gutin, Arash Rafiey, and Anders Yeo. Minimum cost and list homomorphisms to semicomplete digraphs. *Discrete Applied Mathematics*, 154(6):890–897, 2006.

[64] Gregory Gutin, Arash Rafiey, Anders Yeo, and Michael Tso. Level of repair analysis and minimum cost homomorphisms of graphs. *Discrete Applied Mathematics*, 154(6):881–889, 2006.

[65] Michel Habib and Christophe Paul. A survey of the algorithmic aspects of modular decomposition. *Computer Science Review*, 4(1):41–59, 2010.

[66] Magnús M. Halldórsson, Guy Kortsarz, and Hadas Shachnai. Minimizing average completion of dedicated tasks and interval graphs. In *RANDOM-APPROX*, pages 114–126, 2001.

[67] P Hell. Graph partitions with prescribed patterns, 2012.

[68] Pavol Hell. *Retractions de graphes*. PhD thesis, Universite de Montreal, Montreal, Canada, 1972.

[69] Pavol Hell, Miki Hermann, and Mayssam Mohammadi Nevisi. Counting partitions of graphs. *Lecture Notes in Computer Science*, pages 227–236, 2012.

[70] Pavol Hell and Jing Huang. Certifying lexbfs recognition algorithms for proper interval graphs and proper interval bigraphs. *SIAM J. Discrete Math.*, 18(3):554–570, 2004.

[71] Pavol Hell and Jing Huang. Interval bigraphs and circular arc graphs. *Journal of Graph Theory*, 46(4):313–327, 2004.

[72] Pavol Hell, Sulamita Klein, Fábio Protti, and Loana Tito Nogueira. On generalized split graphs. *Electronic Notes in Discrete Mathematics*, 7:98–101, 2001.

[73] Pavol Hell, Monaldo Mastrolilli, Mayssam Mohammadi Nevisi, and Arash Rafiey. Approximation of minimum cost homomorphisms. *Lecture Notes in Computer Science*, page 587598, 2012.

[74] Pavol Hell, Monaldo Mastrolilli, Mayssam Mohammadi Nevisi, and Arash Rafiey. Approximation of minimum cost homomorphisms. In *Lecture Notes in Computer Science*, volume 7501, pages 587–598, 2012.

[75] Pavol Hell and Jaroslav Nesetril. On the complexity of H-coloring. *J. Comb. Theory, Ser. B*, 48(1):92–110, 1990.

[76] Pavol Hell and Jaroslav Nesetril. *Counting list homomorphisms for graphs with bounded degrees in* ***Graphs, Morphisms and Statistical Physics***, volume 63 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 105 – 112. Amer Mathematical Society, 2004.

[77] Pavol Hell and Jaroslav Nesetril. *Graphs and Homomorphisms (Oxford Lecture Series in Mathematics and Its Applications)*. Oxford University Press, USA, 2004.

[78] Pavol Hell and Jaroslav Nesetril. Colouring, constraint satisfaction, and complexity. *Computer Science Review*, 2(3):143–163, 2008.

[79] Pavol Hell and Arash Rafiey. The dichotomy of list homomorphisms for digraphs. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1703–1713. SIAM, 2011.

[80] Pavol Hell and Xuding Zhu. Homomorphisms to oriented paths. *Discrete Mathematics*, 132(1-3):107–114, 1994.

[81] M. Hujter and Zs. Tuza. Precoloring extension. ii. graphs classes related to bipartite graphs. *ACTA MATHEMATICA UNIVERSITATIS COMENIANAE*, 62(1):1–11, 1993.

[82] Mihály Hujter and Zsolt Tuza. Precoloring extension 3: Classes of perfect graphs. *Combinatorics, Probability & Computing*, 5:35–56, 1996.

[83] L. O. James, R. G. Stanton, and D. D. Cowan. Graph decomposition for undirected graphs. In *Proceedings of the Third Southeastern Conference on Combinatorics, Graph Theory, and Computing (Florida Atlantic Univ., Boca Raton, Fla., 1972)*, pages 281–290, Boca Raton, Fla., 1972. Florida Atlantic Univ.

[84] Klaus Jansen. The optimum cost chromatic partition problem. In *CIAC*, pages 25–36, 1997.

[85] Klaus Jansen. Approximation results for the optimum cost chromatic partition problem. *J. Algorithms*, 34(1):54–89, 2000.

[86] Peter Jeavons. On the algebraic structure of combinatorial problems. *Theor. Comput. Sci.*, 200(1-2):185–204, 1998.

[87] Tommy R Jensen and Bjarne Toft. *Graph coloring problems*, volume 39. John Wiley & Sons, 2011.

[88] Tao Jiang and Douglas B. West. Coloring of trees with minimum sum of colors. *Journal of Graph Theory*, 32(4):354–358, 1999.

[89] William T. Trotter Jr. and John I. Moore Jr. Characterization problems for graphs, partially ordered sets, lattices, and families of sets. *Discrete Mathematics*, 16(4):361 – 381, 1976.

[90] Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103, 1972.

[91] Sandi Klavzar. Absolute retracts of split graphs. *Discrete Mathematics*, 134(1-3):75–84, 1994.

[92] Ton Kloks and Yue-Li Wang. On retracts, absolute retracts, and folds in cographs. In *WG*, pages 321–332, 2013.

[93] Norbert Korte and Rolf H. Möhring. An incremental linear-time algorithm for recognizing interval graphs. *SIAM J. Comput.*, 18(1):68–81, 1989.

[94] Leo G. Kroon, Arunabha Sen, Haiyong Deng, and Asim Roy. The optimal cost chromatic partition problem for trees and interval graphs. In *WG*, pages 279–292, 1996.

[95] Ewa Kubicka and Allen J. Schwenk. An introduction to chromatic sums. In *ACM Conference on Computer Science*, pages 39–45, 1989.

[96] Richard E. Ladner. On the structure of polynomial time reducibility. *J. ACM*, 22(1):155–171, 1975.

[97] Benoit Larose and Adrien Lemaître. List-homomorphism problems on graphs and arc consistency. In *ISMVL*, pages 343–348, 2012.

[98] C. G. Lekkerkerker and J. Ch. Boland. Representation of a finite graph by a set of intervals on the real line. 1962.

[99] Nathan Linial. Hard enumeration problems in geometry and combinatorics. *SIAM Journal on Algebraic Discrete Methods*, 7(2):331–335, 1986.

[100] Jiping Liu and Huishan Zhou. Maximum induced matchings in graphs. *Discrete Mathematics*, 170(1-3):277–281, 1997.

[101] Cynthia Loten. Absolute retracts and varieties generated by chordal graphs. *Discrete Mathematics*, 310(10-11):1507–1519, 2010.

[102] Barnaby Martin and Daniël Paulusma. The computational complexity of disconnected cut and 2k 2-partition. In *CP*, pages 561–575, 2011.

[103] Dániel Marx. Precoloring extension on unit interval graphs. *Discrete Applied Mathematics*, 154(6):995–1002, 2006.

[104] Dniel Marx. Precoloring extension on chordal graphs. In Adrian Bondy, Jean Fonlupt, Jean-Luc Fouquet, Jean-Claude Fournier, and JorgeL. Ramrez Alfonsn, editors, *Graph Theory in Paris*, Trends in Mathematics, pages 255–270. Birkhuser Basel, 2007.

[105] Monaldo Mastrolilli and Arash Rafiey. On the approximation of minimum cost homomorphism to bipartite graphs. *Discrete Applied Mathematics*, 161(4-5):670–676, 2013.

[106] Hermann A. Maurer, Ivan Hal Sudborough, and Emo Welzl. On the complexity of the general coloring problem. *Information and Control*, 51(2):128–145, 1981.

[107] Ross M. McConnell. Linear-time recognition of circular-arc graphs. *Algorithmica*, 37(2):93–147, 2003.

[108] Ross M. McConnell and Jeremy Spinrad. Linear-time modular decomposition and efficient transitive orientation of comparability graphs. In *SODA*, pages 536–545, 1994.

[109] Ross M. McConnell and Jeremy Spinrad. Modular decomposition and transitive orientation. *Discrete Mathematics*, 201(1-3):189–241, 1999.

[110] Haiko" "MÏler. Recognizing interval digraphs and interval bigraphs in polynomial time. *Discrete Applied Mathematics*, 78(1-3):189–205, Oct 1997.

[111] Erwin Pesch. Minimal extensions of graphs to absolute retracts. *Journal of Graph Theory*, 11(4):585–598, 1987.

[112] Erwin Pesch. Products of absolute retracts. *Discrete Mathematics*, 69(2):179–188, 1988.

[113] Erwin Pesch and Werner Poguntke. A characterization of absolute retracts of n-chromatic graphs. *Discrete Mathematics*, 57(1-2):99–104, 1985.

[114] J. Scott Provan and Michael O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM J. Comput.*, 12(4):777–788, 1983.

[115] Anja Pruchnewski and Margit Voigt. Precoloring extension for K_4-minor-free graphs. *Journal of Graph Theory*, 60(4):284–294, 2009.

[116] Arash Rafiey and Pavol Hell. Duality for min-max orderings and dichotomy for min cost homomorphisms. *arXiv preprint arXiv:0907.3016*, 2009.

[117] Donald J. Rose and R. Endre Tarjan. Algorithmic aspects of vertex elimination. In *Proceedings of Seventh Annual ACM Symposium on Theory of Computing*, STOC '75, pages 245–254, New York, NY, USA, 1975. ACM.

[118] Mark H. Siggers. A new proof of the H-coloring dichotomy. *SIAM Journal on Discrete Mathematics*, 23(4):2204–2210, Jan 2010.

[119] Jeremy Spinrad, Andreas Brandstädt, and Lorna Stewart. Bipartite permutation graphs. *Discrete Appl. Math.*, 18:279–292, 1987.

[120] Kenneth J. Supowit. Finding a maximum planar subset of a set of nets in a channel. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 6(1):93–94, 1987.

[121] Robert Endre Tarjan. Decomposition by clique separators. *Discrete Mathematics*, 55(2):221–232, 1985.

[122] Robert Endre Tarjan and Mihalis Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 13(3):566–579, 1984.

[123] Robert Endre Tarjan and Mihalis Yannakakis. Addendum: Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 14(1):254–255, 1985.

[124] Marc Tedder, Derek G. Corneil, Michel Habib, and Christophe Paul. Simpler linear-time modular decomposition via recursive factorizing permutations. In *ICALP (1)*, pages 634–645, 2008.

[125] Alan Tucker. Structure theorems for some circular-arc graphs. *Discrete Mathematics*, 7(12):167 – 195, 1974.

[126] Zsolt Tuza. Graph colorings with local constraints - a survey. *Discussiones Mathematicae Graph Theory*, 17(2):161–228, 1997.

[127] Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.

[128] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.

[129] Narayan Vikas. Compaction, retraction, and constraint satisfaction. *SIAM J. Comput.*, 33(4):761–782, 2004.

[130] Narayan Vikas. Computational complexity classification of partition under compaction and retraction. In *COCOON*, pages 380–391, 2004.

[131] Narayan Vikas. A complete and equal computational complexity classification of compaction and retraction to all graphs with at most four vertices and some general results. *J. Comput. Syst. Sci.*, 71(4):406–439, 2005.

[132] Vadim G. Vizing. Coloring the vertices of a graph in prescribed colors. *Diskret. Analiz.*, 29:3–10, 1976.

[133] Margit Voigt. Precoloring extension for 2-connected graphs. *SIAM J. Discrete Math.*, 21(1):258–263, 2007.

[134] Margit Voigt. Precoloring extension for 2-connected graphs with maximum degree three. *Discrete Mathematics*, 309(15):4926–4930, 2009.

[135] Gerd Wegner. *Eigenschaften der Nerven homologisch-einfacher Familien im Rn.* PhD thesis, Universitat Gottingen, Gottingen, Germany, 1967.

Index