Polynomial and FPT algorithms for Chinese Postman, Packing and Acyclicity

Bin Sheng

Supervisor: Prof. Gregory Gutin

Co-Supervisor: Dr Magnus Wahlström

Department of Computer Science
Royal Holloway, University of London

This dissertation is submitted for the degree of
Doctor of Philosophy

RHUL July 2017

I would like to dedicate this thesis to
my loving parents and my wife ...

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university.

Bin Sheng
July 2017

Acknowledgements

First of all, I would like to say thank you to my supervisors, Gregory Gutin and Magnus Wahlström, who helped me enormously during the past four years. Gregory, you are spectacular supervisor, always there to support me. I learned a lot from you, both on academic and personal issues. And I would like to thank Magnus for always willing to answer my questions patiently. I also want to thank my other coauthors, Mark Jones, Anders Yeo, and Florian Barbero, it is a great joy working with you.

My thanks also go to my former teachers, who inspired and encouraged me to pursuit this academic path. Among the others, I want to give my special thanks to my supervisor in master degree Professor Changhong Lu, and Mr Ling Kong for helping me with the application of the CSC Scholarship.

I would like to thank all my friends, in China and in UK. Special thanks go to my friends for playing basketball, football, and badminton with me during my PhD.

I want to deliver my deep gratitude to my family, ever since middle school, I have been studying away. Thanks to my brother and sisters for staying with our parents and grandparents. In paticular, I want to say thank you to my wonderful wife, Jiping. Your support and love means so much to me.

Finally, I would like to point out that my research is supported by China Scholarship Council, and I am grateful for their support.

Abstract

Parameterized algorithms is a new approach to tackle NP-hard problems. Parameterized complexity studies classic hard problems from a multivariate perspective. Apart from the input instance size, it uses some other parameter reflecting useful problem structure. The main purpose is to study how the additional structural information helps to solve the problem. In classic complexity theory, we want to decide whether a problem can be solved in polynomial time, likewise, in parameterized complexity research, we aim to provide fixed parameterized algorithms. In this thesis, we study several problems in graphs for their fixed parameter tractability or polynomial time solvability.

We first study several variants of the Chinese Postman Problem, in which we are asked to find a minimum weight closed walk that traverses each edge or arc at least once. The variants we study include the Directed k-Chinese Postman Problem and the Mixed Chinese Postman Problem. We show that both problems are fixed parameter tractable with the corresponding parameters.

Kernelization is an important subfield in the study of parameterized tractability. It asks whether we can reduce a given problem instance into an equivalent instance of bounded size with respect to the parameter. Polynomial size kernel is of particular interest, as they extract exactly the small hard part of the problem.

In the second part of this thesis, we obtain some linear kernelization results. Let c,k be two positive integers, given a graph $G = (V,E)$, the c-LOAD COLORING problem asks whether there is a c-coloring $\phi : V \to [c]$ such that for every $i \in [c]$, there are at least k edges with both end-vertices colored i. We show that the c-LOAD COLORING problem parameterized by k admits a fixed parameter algorithm, by giving both a linear-vertex and a linear-edge kernel. For a related problem, the star packing problem, we show that it also admits a linear kernel in graphs with no long induced paths. We also studied some problems on edge colored graphs from classic algorithmic perspective, including Odd Properly Colored Cycle detection, Chinese Postman Problem in edge colored graphs, cycles and acyclicity.

Table of contents

Chapter 1

Introduction

This thesis studies polynomial-time and fixed-parameter tractable algorithms. In section 1.1, we first give a sketchy overview of the classical complexity theory. In section 1.2, we describe some of the key concepts in parameterized complexity. Section 1.3 introduces two main tools we use to obtain fixed-parameter tractable algorithms in this thesis. And in section 1.6, we outline our main results and the structure of the rest of the thesis. Section 1.4 lists most of the terminology and notation used in this thesis.

1.1 Classical Complexity Theory

This section is to briefly recall main concepts in classical complexity theory. For a more systematic introduction to classical complexity theory, we refer the readers to the classic book *Computers and Intractability: A Guide to the Theory of NP-completeness* by Garey and Johnson [37].

Computational complexity theory is about classifying the hardness to solve a given problem. Hardness measures the amount of time or memory resources that are required for computation to solve the problem. One of the central topics in classical complexity theory, is whether a problem can be solved in polynomial time with respect to the size of the problem instance.

Turing machine is one of the most widely used mathematical computation model in the study of computational complexity. Recall that a *deterministic* Turing machine is one that for every possible configuration, there is at most one next configuration, while a *non-deterministic* Turing machine can potentially have more than one next configuration for each configuration.

Let Σ denote some finite alphabet set, and Σ^* be the set of all finite strings over Σ. A *problem* (or a *language*) is a subset L of Σ^*. We say a problem L is polynomial-time solvable,

if there exists an algorithm A which takes a string $X \in \Sigma^*$ as an argument, such that running the algorithm A on X takes at most $|X|^c$ steps for some constant c, and $A(X)$ returns Yes if and only if $X \in L$. The class of polynomial time solvable problems is denoted by P, i.e. those problems that can be solved in polynomial time with a *deterministic* Turing machine.

Analogously, the class of problems that can be solved in polynomial time with a *non-deterministic* Turing machine is denoted by NP. A problem L is *NP-hard* if for any problem $L' \in NP$, there is a polynomial reduction from L' to L. A problem L is called *NP-complete* if L is *NP-hard* and $L \in NP$.

The question whether $P = NP$ remains to be the most important open problem in computational complexity theory. Although innumerable researchers have put great effort trying to answer this question for the past decades, the problem has resisted all the attacks. Despite the hardness of the problems, we have to accept the challenge, so what should we do if the problem under consideration is *NP*-complete? Existing options include randomized algorithms, approximation algorithms, heuristic algorithms and so on. Since the late 1980s, we have one more alternative option: fixed-parameter tractable algorithms.

1.2 Introduction to Parameterized Tractability

Researchers showed that many problems are *NP-complete*. In classical complexity theory, all *NP-complete* are regarded as "intractable" uniformly. The practical computational hardness of these hard problems can actually be quite different. People succeeded in designing heuristic algorithms that work rather well for some of them, regardless of their *NP*-hardness. In practice, many real-world instances of *NP*-complete problems can be solved in a reasonable running time.

This phenomenon not only indicates that the *NP*-hardness of a problem should not stop us from trying to solve it, it also poses challenging and intriguing questions for algorithm analysis. What makes an *NP*-complete problem easier to solve? Even further, what lies behind the different layers of hardness for *NP*-hard problems?

A partial answer to this is that hardness results are based on worst-case analysis. The practical success of the algorithms relies on the fact that the worst cases do not show up so often in real life applications. But we also have to admit that the classical single dimensional complexity characterization is rather coarse. It is reasonable to believe that if we take more parameters (not merely the instance size) into consideration, we might be able to design more efficient algorithms. This view motivates researchers to use multivariate algorithmic analysis, with the hope that the parameter we introduce can provide us additional properties.

This is the theoretic motivation to study hard problems in a multivariate framework, i.e., it enables us to give a more precise characterization of the hardness to solve the problem.

The multivariate analysis of the hard problems also has its gain over classical complexity theory from the pragmatic perspective. For many *NP*-complete problems, real-life instances often come with some inherent structures. It would be a great waste to ignore these structures as we do in classical complexity research. This gives us another reason to study fixed-parameter algorithms.

Both the theoretical and practical advantages over classical complexity theory motivate the study of fixed-parameter tractability, in which the additional structure taken into consideration is represented by some parameter k, which can be the sum of several parameters. This section only gives a basic introduction to Parameterized Complexity. For a more detailed introduction, we refer the readers to [29, 35, 70].

Roughly speaking, the goal to study a problem from parameterized perspective is to find some algorithm which runs in time $f(k)poly(|X|)$, where $f(k)$ is some computable function that only depends on k, and $poly(|X|)$ is some polynomial function of $|X|$. Although function f may be fast growing, the running time is still acceptable, if the parameter k we choose is small.

Now we give the formal definition of fixed-parameter tractability.

Definition 1.2.1 *A parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$ is fixed-parameter tractable (FPT) if there exists an algorithm which, for any instance $(X,k) \in \Sigma^* \times \mathbb{N}$, decides whether $(X,k) \in L$ in at most $f(k)|X|^c$ steps, for a constant c and a computable function f that only depends on k.*

In the above definition, the second element k of the problem instance is called the *parameter*.

Since its emergence in the late 1980s, the research in this field has been ever increasing and there are already a lot of related papers published. The chart in Fig 1.1 was produced by Bart M. P. Jansen, *Eindhoven Univ of Technology* in 2016. It shows the number of publications on parameterized complexity and algorithms that Google Scholar recognizes each year from 1995 to 2015. It clearly reveals a huge publication growth over the past decade.

Although the study of FPT has seen a quite astonishing success, as pointed out in [29], "Much remained to be explored!". For instance, the research in this field is mainly theoretical analysis, actual implementation of the FPT algorithms seems to be left in a forgotten corner. Gregory Gutin wrote an article[1] discussing the importance of implementation of fixed-parameter tractable algorithms and he further argued that "getting the hands dirty"

[1]The Parameterized Complexity Newsletter, 2015 December, Vol 11, no 2

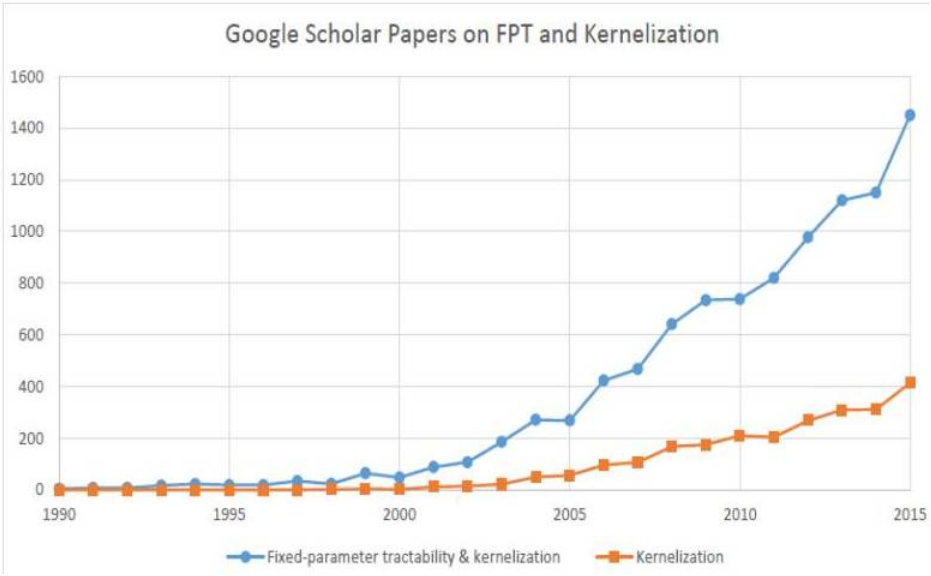

Fig. 1.1 Publication Growth for FPT papers

deserves more appreciation. Fortunately, the first Parameterized Algorithms & Computational Experiments Challenge has successfully been held in the year 2016. It is reasonable to believe people will pay more attention to this line of research in the future.

1.2.1 Fixed-Parameter Intractability

We have seen success of designing fixed-parameterized algorithms for a lot of *NP*-hard problems. We have also came across problems for which it is hard to design FPT algorithms. Actually, there are good explanations for this difficulty. Similarly to classical complexity theory, people have established hardness theory in the FPT area, that is, we can prove some problems are not likely to be fixed-parameterized tractable, under the Exponential Time Hypothesis and other complexity hypotheses. As there are not many results of fixed-parameter intractability in this thesis, we do not go into detailed discussion of this topic, interested reader can easily find related material in books like [29].

1.3 Tools in Parameterized Tractability

In this section, we give a brief introduction to two main techniques people use to obtain positive results in FPT research.

1.3.1 Kernelization

Preprocessing is probably the first step to take in almost all real-world problem solving strategies. In any scenario, one would always like to shrink off the unimportant part in the problem to make the task smaller and easier.

This idea turns out to be also effective in the study of Parameterized Tractability. In FPT community, this preprocessing procedure is named *kernelization*, i.e. the process of pinning down the key part of the problem hardness. We give its formal mathematical definition here.

Definition 1.3.1 *(kernelization) [29] Let $L \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized language. A reduction to a problem kernel, or kernelization, replaces an instance (I,k) by a reduced instance (I',k'), called a problem kernel, such that*

1. *$k' \leq g'(k)$,*
2. *$|I'| \leq g(k)$, for some function g depending only on k, and*
3. *$(I,k) \in L$ if and only if $(I',k') \in L$.*

The reduction from (I,k) to (I',k') must be computable in time polynomial in $|I|+k$.

It is well known that a parameterized problem is fixed-parameter tractable if and only if it admits a kernelization. We measure the effectiveness of the preprocessing procedure by the size of the kernel we get. In particular, we are interested in kernels of polynomially bounded size with respect to the parameter.

In this thesis, we obtain some positive results on polynomial kernelization, see the two sections in Chapter 3.

1.3.2 Treewidth and Dynamic Programming

The notion of tree decomposition and treewidth were first introduced by Robertson and Seymour in [82].

Definition 1.3.2 *Given an undirected graph $G = (V,E)$, a* tree decomposition *of G is a pair $(\mathscr{T},\beta)$, where $\mathscr{T}$ is a tree and $\beta : V(\mathscr{T}) \to 2^V$ such that*

1. *$\bigcup_{x \in V(\mathscr{T})} \beta(x) = V$;*
2. *for each edge $uv \in E$, there exists a node $x \in V(\mathscr{T})$ such that $u,v \in \beta(x)$; and*
3. *for each $v \in V$, the set $\beta^{-1}(v)$ of nodes form a connected subgraph in $\mathscr{T}$.*

The width *of* $(\mathscr{T},\beta)$ *is* $\max_{x\in V(\mathscr{T})}(|\beta(x)|-1)$. *The* treewidth *of G (denoted* $tw(G)$*) is the minimum width of all tree decompositions of G.*

Treewidth is a measure of the similarity of a given graph with a tree. As trees are highly structured graphs, many problems can be solved in polynomial time for them. The smaller the treewidth of a graph, the higher similarity it has with a tree, and thus the easier to solve a problem on it. According to the definition of tree decomposition, the vertices in each node bag form a cut set of the original graph, thus to solve many problem, we may proceed the node bags in the tree from leaf to root one by one, which confines the search space to a function of the treewidth and of n. Thus many NP-hard decision or optimization problems are fixed-parameter tractable on graphs that have bounded treewidth. There is a powerful theorem due to Courcelle which explains this for all problems definable in Monadic Second-Order logic[2].

Theorem 1.3.1 (Courcelle's Theorem) *[35] The following problem is fixed parameter tractable:*

p^*-TW-MC(MSO)
Instance: *a structure A and an MSO-setence* ϕ.
Parameter: $tw(A)+|\phi|$.
Output: *Decide whether* $A \models \phi$

Moreover, there is a computable function f *and an algorithm that solves it in time* $f(k,l)|A|+O(|A|)$*, where* $k := tw(A)$ *and* $l = |\phi|$.

We know that tree decomposition is a very useful structure for algorithm design, to make use of it, we need a way to find such good tree decompositions. But computing an optimal tree decomposition and deciding the treewidth turns out to be NP-complete [6]. The good news is it has been proved to be fixed-parameter tractable parameterized with the treewidth in [16].

Theorem 1.3.2 (Bodlaender's Theorem) *[16] There is a polynomial p and an algorithm that, given a graph* $G=(V,E)$*, computes a tree decomposition of G of width* $k := tw(G)$ *in time at most* $2^{p(k)}n$*, where n is the number of vertices in G.*

There is a faster approximation algorithm of calculating the treewidth.

Theorem 1.3.3 *[35] There is an algorithm that, given a graph* $G=(V,E)$*, computes a tree decomposition of G of width at most* $4tw(G)+1$ *in time* $2^{O(k)}n^2$*, where* $k := tw(G)$ *and* $n=|V|$.

In Chapter 2, we obtained two FPT results using Dynamic Programming based on bounded treewidth and cutwidth.

[2] we refer interested reader for logic complexity to the book [35]

1.4 Terminology and Notation

1.4.1 Graphs

We provide here most of the common terminology and notation used in this thesis. For most of the concepts in undirected graphs, we follow the definitions in the book *Modern Graph Theory* by B. Bolobás [18].

An undirected graph is denoted by an ordered pair $G = (V,E)$, where E is a set of unordered pairs of elements in V. The elements of V are the *vertices* of G and the elements of E are the *edges* of G. Two vertices $u,v \in V$ are adjacent if $\{u,v\} \in E$, i.e. there is an edge between u and v. An edge $\{u,v\}$ is normally written uv for short, thus u,v are *adjacent* if and only if $uv \in E$. And in this case we say vertex u is *incident* with the edge uv, the vertices u and v are referred to as the endpoints of the edge uv. An edge $uu \in E(G)$ would be called a loop. A graph is a *multigraph* if there is more than one copy of edges between some pair of vertices. A graph G is called *simple* if it is not a multigraph and there is no loop in it. In this thesis, when we say a graph we mean a simple undirected graph, unless otherwise stated.

A graph $H = (V',E')$ is a subgraph of G if $V' \subseteq V$ and $E' \subseteq E$. $H = (V',E')$ is an *induced subgraph* of G if $V' \subseteq V$ and E' is the restriction of E to V', i.e. $E' = \{uv \in E : u \in V', v \in V'\}$. For a set of vertices $X \subseteq V(G)$, we use $G[X]$ to denote the induced subgraph of G with vertex set X. $E(X) = E(G[X])$ and $E(X,Y) = \{xy \in E(G) :\ x \in X, y \in Y\}$.

Given a vertex u and a subgraph H of G, we define $N_H(u) = \{v \in V(H) : uv \in E(H)\}$ and $N_H[u] = N_H(u) \cup u$. For a set of vertices X, we define $N_H[X] = \cup_{u \in X} N_H[u]$ and $N_H(X) = \cup_{u \in X} N_H(u) \setminus X$.

For a vertex u in a graph G, we define $N(u) = N_G(u)$ and $N[u] = N_G[u]$, and call $N(u)$ the open neighborhood and $N[u]$ the closed neighborhood of u. For a set of vertices X, the open neighborhood $N(X)$ and closed neighborhood $N[X]$ of X are defined similarly. When we refer to a neighborhood, we mean the open neighborhood unless otherwise specified. We define the degree of u to be the integer $d(u) = |N(u)|$, that is the number of edges incident with vertex u. We use $\Delta(G)$ to denote the maximum degree of G and n its number of vertices.

A *walk* in a multigraph is a sequence $W = v_1 e_1 v_2 \ldots v_{p-1} e_{p-1} v_p$ of alternating vertices and edges such that vertices v_i and v_{i+1} are end-vertices of edge e_i for every $i \in [p-1]$. A walk W is *closed* (*open*, respectively) if $v_1 = v_p$ ($v_1 \neq v_p$, respectively). A *trail* is a walk in which all edges are distinct, a *path* is a non-closed walk in which all vertices are distinct, and a *cycle* is a closed walk where all vertices apart from the first and last ones are distinct. The length of a path (cycle) is the number of edges in it.

A non-empty graph G is called *connected* if there is a path between any two of its vertices. A forest is a graph which contains no cycles. A tree is a connected forest. A vertex in a tree

with degree 1 is called a leaf. An *r-star* is a tree with $r+1$ vertices and r leaves, denoted by $K_{1,r}$.

For two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, the *disjoint union* of G_1 and G_2 is the graph with vertex set $V_1 \cup V_2$ and edge set $E_1 \cup E_2$. An undirected graph is *Eulerian* if and only if it is connected and every vertex has even degree.

An edge $uv \in E(G)$ is called a *bridge* if $G - uv$ contains more connected components than G.

1.4.2 Directed Graphs

For notation and terminology in directed graphs, we follow the book [9]. A directed graph is an ordered pair $D = (V, A)$, where A is a set of ordered pairs of elements from V. The elements of V are the vertices of D and the elements of A are the arcs of D. Similar to undirected graphs, we will use $V(D)$ and $A(D)$ to denote the vertices and arcs of D, respectively. We say an arc (u, v) is an arc from u to v and we normally write it uv. The notions of subgraph and induced subgraph are defined analogously to the undirected case. The *underlying graph* of D is the undirected graph G, where $V(G) = V(D)$, and there is an edge between two vertices u, v in G if and only if $uv \in A(D)$ or $vu \in A(D)$.

The open (closed) neighborhood of a vertex u in D is the neighborhood of u in the underlying graph, and the open (closed) neighborhood of a set of vertices in D is defined similarly. The degree of a vertex u is the degree of u in the underlying graph. The out-neighborhood of u is $N^+(u) = \{v \in V : uv \in A\}$ and its in-neighborhood if $N^-(u) = \{v \in V : vu \in A\}$. The out-degree of u is $d^+(u) = |N^+(u)|$ and the in-degree is $d^-(u) = |N^-(u)|$. For a set S of vertices in D, $N^+(S) = (\cup_{u \in S} N^+(u)) \setminus S$ and $N^-(S) = (\cup_{u \in S} N^-(u)) \setminus S$. We say there is a directed path from a to b in D if D contains a subgraph with vertices $\{a = u_1, \ldots, u_{r+1} = b\}$ and arcs $\{u_i u_i + 1 : 1 \leq i \leq r\}$. We say D has a directed cycle if D contains a subgraph with vertices $\{u_1, \ldots, u_{r+1}\}$ and arcs $\{u_i u_i + 1 : 1 \leq i \leq r\} \cup \{u_{r+1} u_1\}$. A directed graph is acyclic if it does not have a directed cycle.

A directed graph is connected if its underlying graph is connected. A directed graph is strongly connected if for every pair of vertices u, v, there is a directed path from u to v.

Let D be a digraph. For a vertex ordering $\theta = v_1, v_2, \ldots, v_n$ of $V(D)$, the *cutwidth* of θ is the maximum number of arcs between $\{v_1, \ldots, v_i\}$ and $\{v_{i+1}, \ldots v_n\}$ over all $i \in [n]$. The *cutwidth* of D is the minimum cutwidth of all vertex orderings of $V(D)$.

In this thesis, all walks and cycles in directed multigraphs are directed. When we say an integer i is positive, we mean it is $i > 0$. For a positive integer p, $[p]$ will denote the set $\{1, 2, \ldots, p\}$. For integers $a \leq b$, $[a, b]$ will denote the set $\{a, a+1, \ldots, b\}$. Given a directed graph D, a *feedback vertex set* for D is a set S of vertices such that $D - S$ contains no directed

cycles. A *feedback arc set* for D is a set F of arcs such that $D-F$ contains no directed cycles. A vertex v of a digraph is *balanced* if the in-degree of v equals its out-degree. We call $im(u) = d^+(u) - d^-(u)$ the *imbalance* of u. A digraph D is *balanced* if every vertex of D is balanced. A directed graph is Eulerian if and only if it is connected and balanced [9].

For a directed multigraph D, let $\mu_D(xy)$ denote the *multiplicity* of an arc xy of D, which is the number of appearances of it in D. The *multiplicity* of D (denoted by $\mu(D)$) is the maximum of the multiplicities of its arcs.

In this thesis, we also consider mixed graphs, in which we allow both undirected and directed edges. The degree of a vertex in a mixed graph is the number of undirected and directed edges it is adjacent to.

1.4.3 Edge Colored Graphs

A multigraph G is *edge-colored* if each edge of G is assigned a color, and G is called *k-edge-colored* if only colors from $[k] := \{1,2,\ldots,k\}$ are used in the edge coloring of G. Note the edge colorings we consider in this thesis can be arbitrary, not necessarily proper.

A cycle C in an edge-colored graph is *properly colored (PC)* if no pair of incident edges of C have the same color. A walk $W = v_1e_1v_2\ldots v_{p-1}e_{p-1}v_p$ in an edge-colored multigraph is called *properly colored (PC)* if edges e_i and e_{i+1} in W have different colors for every $i \in [p-2]$, and edges e_1 and e_{p-1} have the same color if $v_1 = v_p$. An *Euler trail* in a multigraph G is a closed walk which traverses each edge of G exactly once. And so a *PC Euler trail* in a multigraph G is a properly colored Euler trail.

A vertex v in an edge-colored graph G is called *G-monochromatic*, if all edges incident to v in G are of the same color. Clearly, a PC closed walk has no G-monochromatic vertex.

1.5 Research Background

In this section, we provide some background knowledge for the problems we solve in this thesis.

In Chapter 2, we include several new results related to the classical combinatorial optimization problem, Chinese Postman problem. The Chinese Postman Problem, as suggested by its name, asks to find a best way for a mailman to accomplish his job. It was first raised by the Chinese mathematician MeiGu Guan in [40] around 1960. This problem can be modeled as an optimization problem in graph theory, in the following way.

CHINESE POSTMAN PROBLEM (CPP)	
Input:	A connected edge weighted graph $G = (V, E)$.
Task:	Find a closed walk W of minimum total weight on G such that every edge of G is contained in W.

Since its emergence, this problem has attracted a lot of attention, as it captures a lot of real life situations where the objective is to find the most efficient way of travelling through some network. There are already many papers and theses devoted to postman related problems. In this thesis, we provide three additional results we obtained for it.

The Chinese Postman Problem was shown to be polynomial time solvable, for both undirected and directed graphs [30] (see also [14, 23, 56, 108]).

To solve the Chinese Postman Problem on directed graphs, we can make use of the following minimum-cost circulation problem, which can be solved in polynomial time [96].

MINIMUM-COST CIRCULATION PROBLEM

Instance: A flow network $G = (V, E)$, with vertex set V, edge set E; a lower bound $l(u, v)$ on flow from u to v, a upper bound $u(u, v)$ on flow from u to v, cost $c(u, v)$ per unit flow from u to v.

Output: A flow assignment which minimizes $\Sigma_{(u,v)\in E} c(u, v) f(u, v)$ such that $l(u, v) \leq f(u, v) \leq u(u, v)$ and $\Sigma_{v\in V(G)} f(uv) = \Sigma_{v\in V(G)} f(vu)$, for any $u \in V(G)$.

For a given arc weighted directed multigraph G, we construct a flow network N by assigning lower bound 1, upper bound ∞ and cost $\omega(uv)$ to each arc uv, where $\omega(uv)$ is the weight of uv in G. A minimum-cost circulation in N viewed as an Euler directed multigraph corresponds to a CPP solution and vice versa.

For an undirected multigraph G, we construct an edge-weighted complete graph H whose vertices are the odd degree vertices of G and the weight of an edge xy in H equals the minimum weight path between x and y in G. Now find a minimum-weight perfect matching M in H and add to G a minimum-weight path of G between x and y for each edge xy of M. The resulting Euler multigraph corresponds to a CPP solution on G and vice versa. Note that in the case when there is no odd vertex in G, G itself is the optimal solution for CPP on G.

Although the Chinese Postman Problem is polynomial time solvable for both undirected and directed graphs, we are not as fortunate for mixed graphs. It was proved by Papadimitriou ([72]) in 1976 that Chinese Postman Problem in mixed graphs (MCPP) is NP-hard. In the same paper, Papadimitriou further showed that the NP-completeness holds even for planar graphs in which each vertex has degree at most 3 and each edge or arc has weight at most 1. Interestingly, for mixed graph in which every vertex has even degree, the Chinese Postman

Problem can be solved in polynomial time, this result was obtained by Edmonds and Johnson [30].

There are many other variants of the Chinese Postman Problem, most of which turn out to be NP-hard. For instance, a natural generalization of Chinese Postman Problem arises when we consider asymmetric costs of traversing a street via two different directions, which models the case of blowing wind or a slope. Minieka proposed the Windy Postman Problem modeling for this situation in [68]. The Windy Postman Problem contains MCPP as a special case, so itself is also NP-hard.

The hard variants of Chinese Postman Problem have been studied extensively by approximation or heuristics, there is also a few parameterized complexity study of them. In this thesis, we add two more positive FPT results to this realm, presented in section 2.1 and 2.2. separately.

In Chapter 3, we study two graph packing flavor problems in undirected graphs which were known to be FPT previously. We continue the study mainly from the kernelization perspective, and our goal is to find the smallest possible kernel size.

For a fixed graph H, the problem of deciding whether a graph G has k vertex-disjoint copies of H is called H-PACKING. This problem has many applications (see, e.g., [10, 13, 53]), but unfortunately it is almost always intractable. Indeed, Kirkpatrick and Hell [53] proved that if H contains a component with at least three vertices then H-PACKING is NP-complete. Thus, many approximation, parameterized, and exponential algorithms have been studied for H-PACKING when H is a fixed graph, see, e.g., [10, 32, 33, 80, 104]. The NP-hardness of these problems provides us a good opptunity to apply the techniques in FPT theory. We obtain two linear kernel results for two packing problems on undirected graphs, presented in section 3.1 and 3.2.

Chapter 4 presents the last research theme in this thesis which focuses on the structural properties in edge colored graphs.

In the study of edge colored graphs, many research are conducted under the important notion *properly colored*. Recall that a walk W in an edge-colored multigraph is called properly colored if no two consecutive edges of W have the same color.

PC walks are of interest in many graph theory applications, e.g., in genetic and molecular biology [75, 94, 93], in design of printed circuit and wiring boards [98], and in channel assignment in wireless networks [4, 85]. They are also of interest in graph theory themselves as generalizations of walks in undirected and directed graphs. Indeed, if we assign different colors to all edges of an undirected multigraph, every walk not traversing the same edge twice becomes PC. Also, consider the standard transformation from a directed graph D into a 2-edge-colored graph G by replacing every arc uv of D by a path with a blue edge uw_{uv} and a

red edge $w_{uv}v$, where w_{uv} is a new vertex [9]. Clearly, every directed walk in D corresponds to a PC walk in G (with end-vertices in $V(G)$) and vice versa. There is an extensive literature on PC walks: for a detailed survey of pre-2009 publications, see Chapter 16 of [9], more recent papers include [1, 36, 58–60].

Edge colored graphs are generalization of undirected and directed graphs. We wonder whether existing results on undirected and directed graphs can be extended to edge colored graphs. In Section 2.3, we solved the Chinese Postman Problem on edge colored graphs. In Chapter 4, we continue the study on edge colored graphs. A large portion of current study focus on the existence and detection of certain structure in a given edge colored graph.

Bang-Jensen and Gutin [7] introduced cyclic connectivity as follows. Let $P = \{H_1, \ldots, H_p\}$ be a set of subgraphs of an edge colored multigraph G. The *intersection graph* $\Omega(P)$ of P has the vertex set P and the edge set $\{H_iH_j : V(H_i) \cap V(H_j) \neq \emptyset, 1 \leq i < j \leq p\}$. A pair x, y of vertices in an edge-colored multigraph H is *cyclic connected* if H has a collection of PC cycles $P = \{C_1, \ldots, C_p\}$ such that x and y belong to some cycles in P and $\Omega(P)$ is a connected graph. A maximum cyclic connected induced subgraph of G is called a *cyclic connected component* of G. Note that cyclic connected components partition the vertices of G. Also note that cyclic connectivity for digraphs, where dicycles are considered instead of PC cycles, coincides with strong connectivity.

Aiming for a better understanding of PC connectivity and cycles in edge colored graphs, we introduce 5 types of PC acyclicity, and check whether the Menger's theorem in classic graph theory can be generalized to edge colored graphs.

Theorem 1.5.1 (Menger's Theorem) *[67] Let D be a directed multigraph and let $u, v \in V(D)$ be a pair of distinct vertices. If the arc $uv \notin A(D)$, then the maximum number of internally vertex disjoint (u,v)-paths equals the minimum number of vertices (not including u and v) whose deletion separates u from v.*

1.6 Main Results and Structure of Thesis

We now give a brief introduction to the structure and contents of this thesis. Most of the results in this thesis have been published as conference or journal articles.

In Chapter 2, we study several variants of the classic Chinese Postman Problem.

Section 2.1 based on the paper [43], is devoted to Directed k-Chinese Postman Problem (k-DCPP). In k-DCPP, we are given a connected weighted digraph $G = (V, A)$ and an integer k, and need to find a set of exactly k non-empty closed directed walks (with minimum total weight) such that every arc of G is contained in at least one of the walks. We show that the k-DCPP is fixed-parameter tractable with parameter k.

Section 2.2 which is based on the paper [42], studies the Mixed Chinese Postman Problem (MCPP). In the Mixed Chinese Postman Problem, given an edge-weighted mixed graph G (G may have both edges and arcs), our aim is to find a minimum weight closed walk traversing each edge and arc at least once. The MCPP parameterized with the number of edges was known to be fixed-parameter tractable using a simple argument. Solving an open question raised by van Bevern et al.[102], we prove that the MCPP parameterized with the number of arcs is also fixed-parameter tractable.

Section 2.3 based on the paper [45], considers the Chinese Postman Problem on edge colored graphs. It is well-known that the Chinese Postman Problem is polynomial-time solvable on both undirected and directed graphs. We extend this result to edge-colored multigraphs. Our result is in sharp contrast to the fact that Chinese Postman Problem on mixed graphs is *NP*-hard.

In Chapter 3, we study two problems on undirected graphs from kernelization perspective, both of them have some kind of "graph packing" flavor.

In section 3.1 which is based on the paper [11], we study the c-LOAD COLORING problem. Let c,k be two positive integers, given a graph $G=(V,E)$, the c-LOAD COLORING problem asks whether there is a c-coloring $\varphi : V \to [c]$ such that for every $i \in [c]$, there are at least k edges with both end-vertices colored i. Gutin and Jones studied this problem with $c=2$ in [41]. They proved 2-LOAD COLORING to be fixed-parameter tractable (FPT) with parameter k by obtaining a kernel with at most $7k$ vertices. We extended the result to any fixed c by giving both a linear-vertex and a linear-edge kernel. In the particular case of $c=2$, we obtain a kernel with less than $4k$ vertices and less than $6k+(3+\sqrt{2})\sqrt{k}$+4 edges. These results imply that for any fixed $c \geq 2$, c-LOAD COLORING is FPT and it allows us to design an approximation algorithm with a constant ratio for the optimization version of c-LOAD COLORING (where k is to be maximized).

Section 3.2 based on the paper [12], studies the star packing problem. Let integers $r \geq 2$ and $d \geq 3$ be fixed. Let $\mathscr{G}_d$ be the set of graphs with no induced path on d vertices. We study the problem of packing k ($k \geq 2$) vertex-disjoint copies of $K_{1,r}$ into a graph G from parameterized preprocessing, i.e., kernelization, point of view. We show that every graph $G \in \mathscr{G}_d$ can be reduced, in polynomial time, to a graph $G' \in \mathscr{G}_d$ with $O(k)$ vertices such that G has at least k vertex-disjoint copies of $K_{1,r}$ if and only if G' does.

In Chapter 4, we discuss about Properly Colored (PC) cycles and acyclicity in edge colored graphs.

In section 4.1, which is based on the paper [44], we introduce and study five types of PC acyclicity in edge-colored graphs such that graphs of PC acyclicity of type i is a proper superset of graphs of acyclicity of type $i+1$, $i=1,2,3,4$. The first three types are equivalent

to the absence of PC cycles, PC closed trails, and PC closed walks, respectively. While graphs of types 1, 2 and 3 can be recognized in polynomial time, the problem of recognizing graphs of type 4 is, somewhat surprisingly, *NP*-hard even for 2-edge-colored graphs (i.e., when only two colors are used). The same problem with respect to type 5 is polynomial-time solvable for all edge-colored graphs. Using the five types, we investigate the border between intractability and tractability for the problems of finding the maximum number of internally vertex-disjoint PC paths between two vertices and the minimum number of vertices to meet all PC paths between two vertices.

Section 4.2 is based on the paper [50], in which we consider the problem of detecting odd properly colored cycles in edge colored graphs. It is well-known that an undirected graph has no odd cycle if and only if it is bipartite. A less obvious, but similar result holds for directed graphs: a strongly connected digraph has no odd directed cycle if and only if it is bipartite. We study this problem and show how to decide if there exists an odd PC cycle in a given edge-colored graph. As a by-product, we show how to detect if there is a perfect matching in a graph with even (or odd) number of edges in a given edge set.

Chapter 2

Chinese Postman Problem

2.1 k-DCPP

Let $G = (V,A)$ be a connected digraph, where each arc $a \in A$ is assigned a non-negative integer weight $\omega(a)$ (G is a *weighted digraph*). As we already mentioned, the DIRECTED CHINESE POSTMAN PROBLEM is a well-studied polynomial-time solvable problem in combinatorial optimization [9, 30, 108].

DIRECTED CHINESE POSTMAN PROBLEM (DCPP)	
Input:	A connected weighted digraph $G = (V,A)$.
Task:	Find a closed directed walk T of minimum total weight on G such that every arc of G is contained in T.

In this section, we will investigate the following generalization of DCPP.

DIRECTED k-CHINESE POSTMAN PROBLEM (k-DCPP)	
Input:	A connected weighted digraph $G = (V,A)$ and an integer k.
Task:	Find a set of exactly k non-empty closed directed walks with minimum total weight such that every arc of G is contained in at least one of the walks.

Note that the k-DCPP can be extended to directed multigraphs (that may include parallel arcs but no loops), but the extended version could be reduced to the one on digraphs by subdividing parallel arcs and adjusting weights appropriately. Since it is more convenient, we consider the k-DCPP for digraphs only.

In the literature, the undirected version of k-DCPP, abbreviated k-UCPP, has also been studied. The k-UCPP was formulated to model the scenario with k vehicles, which cooperate to accomplish to task of traversing each edge or arc at least once. If a vertex v of G is part of

the input and we require that each of the k walks contains v then the k-DCPP and k-UCPP are polynomial-time solvable [73, 111]. We just need to add $k - d_{G_T}(v)$ copies of a minimum weight cycle through v, where G_T is an optimal solution for DCPP on G. However, in general the k-DCCP is NP-complete [48], as is the k-UCPP [48, 97].

Lately research in parameterized algorithms and complexity for the CPP and its generalizations was summarized in [102] and reported in [90]. Several recent results described there are of Niedermeier's group who identified a number of practically useful parameters for the CPP and its generalizations, obtained several interesting results and posed some open problems, see, e.g. [28, 91, 92]. Sorge [90] and van Bevern *et al.* [102] suggested to study the k-UCPP as a parameterized problem with parameter k and asked whether the k-UCPP is fixed-parameter tractable, i.e. can be solved by an algorithm of running time $O(f(k)n^{O(1)})$, where f is a function of k only and $n = |V|$.

Theorem 2.1.1 *The k-DCPP is fixed-parameter tractable.*

Our proof is very different from that in Gutin et al. [48] for the k-UCPP. While the latter was based on a simple reduction to a polynomial-size kernel, we give a fixed-parameter algorithm directly using significantly more powerful tools. In particular, we use an *approximation* algorithm of Grohe and Grüber [38] for the problem of finding the maximum number $\nu_0(D)$ of vertex-disjoint directed cycles in a digraph D. Their algorithm is based on the celebrated paper by Reed *et al.* [81] on bounding $\nu_0(D)$ by a function of $\tau_0(D)$, the minimum size of a feedback vertex set of D. We also use the well-known fixed-parameter algorithm of Chen *et al.* [22] for the feedback vertex set problem on digraphs.

We also consider the following well-known problem related to the k-DCPP.

k-ARC-DISJOINT CYCLES PROBLEM (k-ADCP)	
Input:	A digraph D and an integer k.
Task:	Decide whether D has k arc-disjoint directed cycles.

Crucially, we are interested in the k-ADCP because it is related to k-DCPP. Since if we are given a set of k arc-disjoint cycles, we can solve the k-DCPP in polynomial time (see Lemma 2.1.5). However, k-ADCP is important in its own right.

The problem is NP-hard in general but polynomial-time solvable for planar digraphs [64]. In fact, for planar digraphs the maximum number of arc-disjoint directed cycles equals the minimum size of a feedback arc set [9]. It is natural to consider k as the parameter for the k-ADCP. It follows easily from the results of Slivkins [89] that the k-ADCP is W[1]-hard. It remains W[1]-hard for quite restricted classes of directed multigraphs, e.g., for directed multigraphs which become acyclic after deleting two sets of parallel arcs [89]. Here we

show that the k-ADCP-EULER, the k-ADCP on Euler digraphs, is fixed-parameter tractable, generalizing a result in [89] (Theorem 3.1). k-ADCP-EULER was shown to be NP-hard by Vygen [103].

Theorem 2.1.2 *The k-ADCP-EULER is fixed-parameter tractable.*

Interestingly, the problem of deciding whether a digraph has k vertex-disjoint directed cycles, which is W[1]-hard (also easily follows from the results of Slivkins [89]), remains W[1]-hard on Euler digraphs. Indeed, consider a non-Euler digraph D and let $\nu_0(D)$ denote the maximum number of vertex-disjoint directed cycles in D. Construct a new digraph H from D by adding two new vertices x and y, arcs xy and yx and the following extra arcs between x and the vertices of D: for each $v \in V(D)$ add $\max\{d^-(v) - d^+(v), 0\}$ parallel arcs vx and $\max\{d^+(v) - d^-(v), 0\}$ parallel arcs xv, where $d^-(v)$ and $d^+(v)$ are the in-degree and out-degree of v, respectively. To eliminate parallel arcs, it remains to subdivide all arcs between x and $V(D)$. Now it is sufficient to observe that H is Euler and $\nu_0(H) = \nu_0(D) + 1$.

To prove Theorems 2.1.1 and 2.1.2, we study the following problem that generalizes the k-DCPP (in the case when an optimal solution exists in which the number of times each arc visited by each closed walk is restricted) and k-ADCP. Let $b \leq c$ be non-negative integers.

DIRECTED k-WALK $[b,c]$-COVERING PROBLEM ($k[b,c]$-DWCP)	
Input:	A connected weighted digraph $G = (V,A)$ and an integer k.
Task:	Find a set of k non-empty closed directed walks with minimum total weight in which every arc of G appears between b and c times in total.

In Section 2.1.2 we will prove the following theorem.

Theorem 2.1.3 *Let (G,k) be an instance of $k[b,c]$-DWCP and suppose we are given a vertex ordering $\theta = (v_1, v_2, \ldots, v_n)$ of G with cutwidth at most p. Then, in time $O^*((c2^k)^p 2^k)$, we can solve (G,k) and find an optimal feasible solution if one exists.*

In this thesis, the O^* notation omits polynomial function of n. Note that when c and p are upper-bounded by functions of k, the algorithm of this theorem is fixed parameterized.

The contents are organised as follows. In Section 2.1.1, we prove six lemmas providing structural results for the k-DCPP and k-ADCP-EULER, which will later be used to reduce these problems to $k[b,c]$-DWCP. In Section 2.1.2, we prove Theorem 2.1.3. In Section 2.1.3, we put the results of the previous two sections together to prove Theorems 2.1.1 and 2.1.2.

The key results of Section 2.1.1 are as follows. Lemma 2.1.3 shows that, given an Euler directed graph, we can either find k arc-disjoint cycles or a vertex ordering with cutwidth bounded by a function of k. This allows us to either solve the k-ADCP-EULER directly or reduce it to the $k[0,1]$-DWCP on a graph of bounded cutwidth, allowing us to apply Theorem 2.1.3. Lemmas 2.1.5 and 2.1.6 study the Eulerian graph G_T derived from a solution T to the DCPP on G. Lemma 2.1.5 shows that given k arc-disjoint cycles in G_T, we can solve the k-DCPP on G in polynomial time. Lemma 2.1.6 shows that if no arc appears in G_T more than k times (in particular if there are fewer than k arc-disjoint cycles in G_T), there is an optimal solution for the k-DCPP such that no arc is visited more than k times in total by the k walks of the solution. This allows us to reduce the k-DCPP to the $k[1,k]$-DWCP, and Lemma 2.1.3 allows us to bound the cutdwidth of the graph. Thus, in this case we can again apply Theorem 2.1.3.

2.1.1 Structural Results and Fixed-Parameter Algorithms

The next lemma is a simple sufficient condition for an Euler digraph to contain k arc-disjoint cycles.

Lemma 2.1.1 *Every balanced digraph D having a vertex of out-degree at least $k \geq 1$, contains k arc-disjoint cycles that can be found in polynomial time.*

Proof For $k = 1$, it is true as D has a cycle that can be found in polynomial time. Let $k \geq 2$ and let C be a cycle in D. Observe that after deleting the arcs of C, D has a vertex of out-degree at least $k-1$ and we are done by induction hypothesis. ∎

It follows from Reed *et al.* [81] and Propositions 13.3.1 and 15.3.1 in [9] that there is a function $f : \mathbb{N} \to \mathbb{N}$ such that for every k, if a digraph D does not have k arc-disjoint cycles, then it has a feedback arc set with at most $f(k)$ arcs. This result can be easily extended to directed multigraphs; we reduce multigraphs to graphs by subdividing parallel arcs. Using this result, Grohe and Grüber [38] showed that there is a non-decreasing and unbounded function $h : \mathbb{N} \to \mathbb{N}$ and a fixed-parameter algorithm that returns at least $h(k)$ arc-disjoint cycles for a digraph D if D has at least k arc-disjoint cycles (k is the parameter).

Let $h^{-1} : \mathbb{N} \to \mathbb{N}$ be defined by $h^{-1}(q) = \min\{p : h(p) \geq q\}$. Since h is a non-decreasing and unbounded function, h^{-1} is a non-decreasing and unbounded function. Combining the above results, we find that for every digraph D, either the algorithm of Grohe and Grüber returns at least k arc-disjoint cycles, or D has a feedback arc set of size at most $f(h^{-1}(k))$.

Chen *et al.* [22] designed a fixed-parameter algorithm that decides whether a digraph D contains a feedback vertex set of size k (k is the parameter). As this is an iterative

compression algorithm, it can be easily modified to an algorithm for finding a minimum feedback vertex set in D (the running time of the latter algorithm is $q(\tau_0(D))n^{O(1)}$, where $\tau_0(D)$ is the minimum size of a feedback vertex set in D, $n = |V(D)|$ and $q(k) = 4^k k!$). The modified algorithm can be used for finding a minimum feedback arc set in D as the line graph of D, denoted by $H = L(D)$, can be constructed in polynomial time, such that D has a feedback arc set of size k if and only if H has a feedback vertex set of size k, see , e.g., [9] (Proposition 15.3.1).

Lemma 2.1.2 *There is a function $g : \mathbb{N} \to \mathbb{N}$ and a fixed-parameter algorithm such that for a directed multigraph D, the algorithm returns either k arc-disjoint cycles or a feedback arc set of size at most $g(k)$ (here k is the parameter).*

Proof By subdividing arcs, we may assume that D is a digraph, i.e. D has no parallel arcs. Let $\kappa := k-1$ and perform the following loop: for $\kappa := \kappa + 1$ run both Grohe-Grüber algorithm and Chen *et al.* algorithm on D with parameter κ until we get either at least k arc-disjoint cycles or a feedback arc set of size at most κ. Note that by [81], the loop will be completed for $\kappa \leq f(h^{-1}(k))$. Thus, our procedure is a fixed-parameter algorithm with respect to parameter k and we may set $g(k) = f(h^{-1}(k))$. ∎

Lemma 2.1.3 *Let $g : \mathbb{N} \to \mathbb{N}$ be the function in Lemma 2.1.2. Let D be an Euler directed multigraph. We can obtain either k arc-disjoint cycles of D or a vertex ordering of cutwidth at most $2g(k)$.*

Proof Let us run the procedure of Lemma 2.1.2 for D and k. If we get k arc-disjoint cycles, we are done. Otherwise, we get a feedback arc set F of D such that $|F| \leq g(k)$. Then $D' = D - F$ is an acyclic directed multigraph. We let $\theta = (v_1, \ldots, v_n)$ be an acyclic ordering of D', i.e., D' has no arc of the form $v_i v_j$, $i > j$, (it is well-known that such an ordering exists [9]). Now θ is a vertex ordering for D with at most $|F|$ arcs from $\{v_{i+1}, \ldots, v_n\}$ to $\{v_1, \ldots, v_i\}$ for each $i \in [n-1]$, and because D is Euler there are the same number of arcs from $\{v_1, \ldots, v_i\}$ to $\{v_{i+1}, \ldots, v_n\}$ [9, Corollary 1.7.3]. So θ is a vertex ordering with cutwidth at most $2g(k)$. ∎

In the rest of this section, $G = (V, A)$ is a connected weighted directed graph. For a solution $T = \{T_1, \ldots, T_k\}$ to the k-DCPP on G ($k \geq 1$), let $G_T = (V, A_T)$, where A_T is a multiset containing all arcs of A, each as many times as it is traversed in total by $T_1 \cup \cdots \cup T_k$.

Lemmas 2.1.4 and 2.1.5 are similar to two simple results obtained for the k-UCPP in [48]. Note that given k closed walks which cover all the arcs of a digraph, their union forms a closed walk covering all the arcs and, therefore, it is a solution for the DCPP. Hence, the following proposition holds.

Lemma 2.1.4 *The weight of an optimal solution for the k-DCPP on G is not smaller than the weight of an optimal solution for the DCPP on G.*

Lemma 2.1.5 *Let T be an optimal solution for the DCPP on G. If G_T contains at least k arc-disjoint cycles, then the weight of an optimal solution for the k-DCPP on G is equal to the weight of an optimal solution of the DCPP on G. Furthermore if k arc-disjoint cycles in G_T are given, then an optimal solution for the k-DCPP can be found in polynomial time.*

Proof Note that G_T is an Euler directed multigraph and so every vertex of G_T is balanced. Let $\mathscr{C}$ be any collection of k arc-disjoint cycles in G_T. Delete all arcs of $\mathscr{C}$ from G_T and observe that every vertex in the remaining directed multigraph G' is balanced. Find an optimal DCPP solution for every connected component of G' and append each such solution F to a cycle in $\mathscr{C}$ which has a common vertex with F. As a result, in polynomial time, we obtain a collection Q of k closed walks for the k-DCPP on G of the same weight as T. So Q is optimal by Lemma 2.1.4. ∎

Thus, Lemmas 2.1.1 and 2.1.5 implies that if the multiplicity (recall we defined it in **??**) $\mu(G_T) \geq k$ for any optimal solution T of the DCPP on G, then there is an optimal solution of the k-DCPP on G with weight equal to the weight of G_T. The next lemma helps us in the case that $\mu(G_T) \leq k-1$.

Lemma 2.1.6 *Let T be an optimal solution of the DCPP on G such that $\mu(G_T) \leq k-1$. Then there is an optimal solution W for the k-DCPP on G such that $\mu(G_W) \leq k$.*

Proof Let T be an optimal solution of DCPP on G and let $\mu(G_T) \leq k-1$. Suppose that there is an optimal solution W of the k-DCPP on G such that $\mu(G_W) > k$.

Let $\delta(xy) = \mu_{G_W}(xy) - \mu_{G_T}(xy)$ for each arc xy of G. Consider a directed multigraph H' with the same vertex set as G and in which xy is an arc of multiplicity $|\delta(xy)|$ if it is an arc in G and $\delta(xy) \neq 0$. We say that an arc xy of H' is *positive* (*negative*) if $\delta(xy) > 0$ ($\delta(xy) < 0$).

For a digraph D and its vertex x, let $N_D^+(x)$ and $N_D^-(x)$ denote the sets of out-neighbors and in-neighbors of x, respectively. As G_W and G_T are both Euler graphs, we have that

$$\begin{aligned} &\sum_{y \in N_{H'}^+(x)} \delta(xy) - \sum_{y \in N_{H'}^-(x)} \delta(yx) \\ &= \sum_{y \in N_G^+(x)} (\mu_{G_W}(xy) - \mu_{G_T}(xy)) \quad - \quad \sum_{y \in N_G^-(x)} (\mu_{G_W}(yx) - \mu_{G_T}(yx)) = 0 \end{aligned}$$

for each vertex x in G. Now create the directed multigraph H by reversing every negative arc of H' (i.e., replace every negative arc uv by the negative arc vu, keeping the weight of the arcs the same), and observe that H is balanced.

Thus, the arcs of H can be decomposed into a collection $\mathscr{C} = \{C_1, \ldots, C_t\}$ of cycles. We define the weight $\omega(C_i)$ of a cycle C_i of $\mathscr{C}$ as the sum of the weights of its positive arcs minus the sum of the weights of its negative arcs, and assume that $\omega(C_1) \leq \cdots \leq \omega(C_t)$.

Set $F_0 = G_T$ and for $i \in [t]$, construct F_i from F_{i-1} as follows: for each arc xy of C_i, if xy is a positive arc in H add a copy of xy to F_{i-1} and if xy is a negative arc in H remove a copy of yx from F_{i-1}. Thus $F_0, F_1, \ldots, F_t$ is a sequence of graphs with $F_0 = G_T, F_t = G_W$, and F_i is an Euler graph for each $i \in [t]$. Furthermore, the multiplicity of each arc xy changes by at most 1 between F_{i-1} and F_i for each $i \in [t]$, and no arc will have its multiplicity both increase and decrease over the course of $F_0, F_1, \ldots, F_t$. Therefore, every arc uv has multiplicity between $\mu_{G_T}(uv)$ and $\mu_{G_W}(uv)$ in each F_i, and so each F_i is a feasible solution for DCPP on G.

Since T is optimal, $\omega(F_0) \leq \omega(F_1) = \omega(F_0) + \omega(C_1)$ and so $\omega(C_1) \geq 0$. Due to the ordering of cycles of $\mathscr{C}$ according to their weights, $\omega(C_i) \geq 0$ for $i \in [t]$ and so $\omega(F_i) \geq \omega(F_{i-1})$ for $i \in [t]$.

Since $\mu(F_0) \leq k-1$ and $\mu(F_t) > k$, and as the multiplicity of each arc changes by at most 1 each time, there is an index j such that $\mu(F_j) = k$. Then the out-degree of some vertex of F_j is at least k and so by Lemma 2.1.1, F_j has k arc-disjoint cycles. Similarly to Lemma 2.1.5, it is not hard to show that there is a solution U of k-DCPP on G of weight $\omega(F_j)$. Since W is optimal and $\omega(F_j) \leq \omega(F_t) = \omega(G_W)$, U is also optimal and we are done. ∎

2.1.2 Proof of Theorem 2.1.3

Theorem 2.1.3 is proved by providing a dynamic programming (DP) algorithm of required complexity. We first make an observation to simplify the DP algorithm.

Lemma 2.1.7 *Let $G = (V, A)$ define an instance of $k[b, c]$-DWCP. The instance has a solution of weight at most ρ if and only if there exist (not necessarily connected) non-empty directed multigraphs $G_1, \ldots, G_k$ with the following properties:*

- *All multigraphs $G_1, \ldots, G_k$ use only arcs of G (each, possibly, multiple times);*
- *G_1 is a balanced multigraph;*
- *For $2 \leq i \leq k$, G_i is a balanced digraph (with no parallel arcs);*
- *Each arc $a \in A$ occurs between b and c times in the multigraph[1] $G_1 \cup \cdots \cup G_k$, and the total weight of this multigraph is at most ρ.*

[1] Here, as in the proof, the union of multigraphs means that the multiplicity of an arc in the union equals the sum of multiplicities of this arc in the multigraphs of the union.

Proof On the one hand, let $W_1, \ldots, W_k$ be a solution to the $k[b,c]$-DWCP instance of weight at most ρ, where each W_i is a closed directed walk. For each $i \in [k]$, let Q_i be the directed multigraph whose vertices are the vertices visited by W_i and which contains an arc uv of multiplicity μ if uv is traversed exactly μ times by W_i. For each $i \geq 2$, if Q_i has parallel arcs, let G_i be a cycle in Q_i and let $Q_i' = Q_i \setminus A(G_i)$ and, otherwise (i.e., Q_i has no parallel arcs), let $G_i = Q_i$ and let Q_i' be empty. Now let $G_1 = Q_1 \cup Q_2' \cup \cdots \cup Q_k'$. Observe that all properties of the lemma are satisfied.

On the other hand, consider directed multigraphs $G_1, \ldots, G_k$ satisfying the properties of the lemma. If all multigraphs G_i are connected, then they are all Euler. Therefore we can find an Euler tour W_i for each graph G_i, which forms the solution to the $k[b,c]$-DWCP instance. If $b = 0$, then we may replace each graph G_i with a cycle C_i contained in G_i, and produce a solution to $k[b,c]$-DWCP that consists of k (not necessarily pairwise arc-disjoint) cycles.

Finally, if not all multigraphs are connected and $b > 0$, we proceed as follows. First, select for each multigraph G_i, $i > 1$ an arbitrary connected component H_i, and move all other components of G_i to G_1, increasing arc multiplicity as appropriate. Next, as long as G_1 remains unconnected, let H be an arbitrary connected component of G_1. As $b > 0$ and G is connected, some component H_i, $i > 1$ must intersect a vertex of H; we may move H to the multigraph G_i and maintain that G_i is connected. Repeat this until G_1 (and hence each multigraph G_i) is connected. Note that this does not change the arc multiplicity or the weight of the solution. Now each multigraph G_i is Euler, and again we can find a solution. ∎

Our DP algorithm will calculate a function $\Phi : A(G) \times [k] \to [0,c]$ corresponding to an optimal solution to the $k[b,c]$-DWCP on G. More precisely, $\Phi(a,j)$ will be the number of copies of arc a in walk number j, for each $a \in A(G), j \in [k]$. The following definitions and the next lemma allow us to express the result of Lemma 2.1.7 in terms of this function.

Given a set of arcs M and a function $\phi : M \times [k] \to [0,c]$, we say that ϕ is *valid* if for each arc $a \in M$, we have that $\sum_{j \in [k]} \phi(a,j) \in [b,c]$, and $\phi(a,j) \leq 1$ for $2 \leq j \leq k$.

Given a vertex v, we say ϕ is *balanced* for v if for each $j \in [k]$,

$$\sum_{uv \in M} \phi(uv,j) = \sum_{vu \in M} \phi(vu,j)$$

that is, v is a balanced vertex in the directed mulitgraph containing $\phi(a,j)$ copies of each arc a.

Lemma 2.1.8 *Let $G = (V,A)$ define an instance of $k[b,c]$-DWCP. The instance has a solution of weight at most ρ if and only if there exists a function $\Phi : A \times [k] \to [0,c]$ such that*

1. *Φ is valid;*

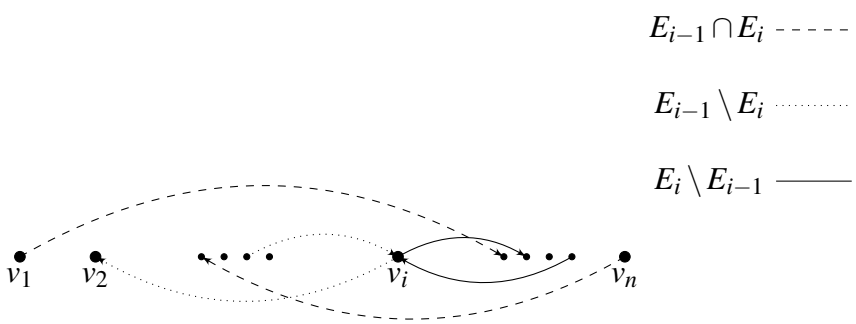

Fig. 2.1 $E_i \setminus E_{i-1}$, $E_{i-1} \setminus E_i$ and $E_{i-1} \cap E_i$

2. Φ *is balanced for each vertex in* V*;*

3. $\sum_{a \in A} \Phi(a, j) > 0$ *for each* $j \in [k]$*; and*

4. $\sum_{j \in [k]} \sum_{a \in A} \Phi(a, j) \cdot \omega(a) \leq \rho$.

Proof Suppose first there is a solution of weight at most ρ, and let $G_1, \ldots, G_k$ be the directed multigraphs given by Lemma 2.1.7. Let $\phi : A \times [k] \to [0, c]$ be the function such that $\phi(a, j)$ is the number of copies of an arc a in the graph G_j. As each arc appears between b and c times in $G_1 \cup \cdots \cup G_k$ and G_j has no parallel arcs for $j \geq 2$, we have that ϕ is valid. As each multigraph G_j is balanced, we have that ϕ is balanced for each vertex. As each multigraph is non-empty, we have that $\sum_{a \in A} \phi(a, j) > 0$ for each $j \in [k]$. Finally, $\sum_{j \in [k]} \sum_{a \in A} \phi(a, j) \cdot \omega(a)$ is exactly the total weight of $G_1 \cup \cdots \cup G_k$, which is at most ρ. Therefore, ϕ satisfies the conditions of the lemma.

Conversely, let $\phi : A \times [k] \to [0, c]$ be a function satisfying the conditions of the lemma, and for each $j \in [k]$, let G_j be the directed multigraph containing $\phi(a, j)$ copies of each arc a. As $\sum_{a \in A} \phi(a, j) > 0$, each multigraph G_j is non-empty. By construction, each multigraph G_j uses only arcs of G. As ϕ is balanced for each vertex, we have that each multigraph G_j is balanced. As ϕ is valid, we have that G_j has no parallel arcs for $j \geq 2$, and each arc $a \in A$ occurs between b and c times in $G_1 \cup \cdots \cup G_k$. Finally, the total weight of $G_1 \cup \cdots \cup G_k$ is $\sum_{j \in [k]} \sum_{a \in A} \phi(a, j) \cdot \omega(a) \leq \rho$. So by Lemma 2.1.7 there is a solution to the $k[b, c]$-DWCP instance of weight at most ρ. ∎

Let $\theta = (v_1, v_2, \ldots, v_n)$ be a vertex ordering of a digraph G of cutwidth at most p. For each $i \in [n-1]$, let E_i be the set of arcs of the form $v_j v_h$ or $v_h v_j$, where $j \leq i$ and $h > i$. In addition let $E_0 = \emptyset$ and $E_n = \emptyset$. As θ has cutwidth at most p, $|E_i| \leq p$ for each i. We refer to $E_0, E_1, \ldots, E_n$ as the *arc bags* of θ. For each $i \in \{0, 1, \ldots, n\}$, let $E_{\leq i} = \bigcup_{0 \leq j \leq i} E_j$. We give an illustration of the difference between E_{i-1} and E_i in Fig 2.1.

We now give an intuitive description of the DP algorithm before giving technical details. Our DP algorithm will process each arc bag of θ in turn, from E_0 to E_n. For each arc bag E_i, we store the weights of a range of partial solutions. A function ϕ is used to represent how many times each arc in the bag E_i is used by each walk in the solution. Finally, a set S provides a guarantee that certain walks are non-empty. This is to ensure we don't produce a solution which uses less than k non-empty walks.

Given $i \in [n]$, a valid function $\phi : E_i \times [k] \to [0,c]$ and a subset S of $[k]$, we define $\chi(E_i, \phi, S)$ to be the minimum integer ρ for which there exists a function $\Phi : E_{\leq i} \times [k] \to [0,c]$ satisfying the following conditions:

1. Φ is valid;
2. Φ extends ϕ, i.e. $\Phi(a,j) = \phi(a,j)$ for each $a \in E_i, j \in [k]$;
3. For each $h \leq i$, Φ is balanced for v_h;
4. $\sum_{a \in E_{\leq i}} \Phi(a,j) > 0$ for each $j \in S$; and
5. $\sum_{j \in [k]} \sum_{a \in E_{\leq i}} \Phi(a,j) \cdot \omega(a) \leq \rho$.

If no such integer ρ exists, then we let $\chi(E_i, \phi, S) = \infty$.

Observe that if Φ is a function satisfying the above conditions, then $\chi(E_i, \phi, S) \leq \rho$. In such a case we will call Φ a *witness* for $\chi(E_i, \phi, S) \leq \rho$. Thus, $\chi(E_i, \phi, S)$ is the minimum ρ such that there exists a witness for $\chi(E_i, \phi, S) \leq \rho$.

Note that if Φ is a witness for $\chi(E_i, \phi, S) \leq \rho$, it may be the case that $\sum_{a \in E_{\leq i}} \Phi(a,j) > 0$ for some $j \notin S$. In particular, any witness for $\chi(E_i, \phi, S) \leq \rho$ is also a witness for $\chi(E_i, \phi, S') \leq \rho$, for any $S' \subseteq S$. This allows us to simplify the recursion step in Lemma 2.1.10.

The next lemma follows from Lemma 2.1.8 and the fact that $E_n = \emptyset$ and $E_{\leq n} = A(G)$.

Lemma 2.1.9 *Let $\phi : E_n \times [k] \to [b,c]$ be the empty function. Then there is a solution for the $k[b,c]$-DCPP on G of weight at most ρ if and only if $\chi(E_n, \phi, [k]) \leq \rho$.*

Lemma 2.1.10 *Consider an arc bag E_i, for $i \geq 1$. Let $E_i^* = E_i \setminus E_{i-1}$. For any valid $\phi : E_i \times [k] \to [0,c]$ and $S \subseteq [k]$, let $Y = \sum_{j \in S} \sum_{a \in E_i^*} \phi(a,j) \cdot \omega(a)$, and let $S' = \{j \in S : \sum_{a \in E_i^*} \phi(a,j) = 0\}$.*

Then the following recursion holds:

$$\chi(E_i, \phi, S) = Y + \min_{\phi'} \chi(E_{i-1}, \phi', S')$$

where the minimum is taken over all valid $\phi' : E_{i-1} \times [k] \to [0,c]$ *satisfying the following conditions:*

- *For all* $a \in E_i \cap E_{i-1}$ *and all* $j \in [k]$, $\phi'(a,j) = \phi(a,j)$*; and*
- *The function* $\phi \cup \phi'$ *is balanced for* v_i.

If there is no ϕ' *satisfying these conditions, then* $\chi(E_i,\phi,S) = \infty$.

Furthermore, if there exists ϕ' *satisfying the above conditions and we are given a witness* Φ' *for* $\chi(E_{i-1},\phi',S') \leq \rho'$*, then we can construct a witness for* $\chi(E_i,\phi,S) \leq Y + \rho'$ *in polynomial time.*

Proof It will be useful to note that for each $h \leq i$, all arcs incident to v_h are contained in $E_{\leq i}$, and that all arcs incident to v_i are contained in $E_{i-1} \cup E_i$.

We first show that $\chi(E_i,\phi,S) \geq Y + \min_{\phi'} \chi(E_{i-1},\phi',S')$. Suppose that $\chi(E_i,\phi,S) = \rho$ and $\rho \neq \infty$, and let $\Phi : E_{\leq i} \times [k] \to [0,c]$ be a witness for $\chi(E_i,\phi,S) \leq \rho$. Let Φ' be Φ restricted to $E_{\leq i-1}$ and let ϕ' be Φ restricted to E_{i-1}. Let $\rho' = \sum_{j\in[k]} \sum_{a\in E_{\leq i-1}} \Phi(a,j)$.

As Φ is valid, so are Φ' and ϕ'. Consider a vertex v_h, $h \leq i-1$. As Φ is balanced for v_h, and all arcs incident to v_h are contained in $E_{\leq h} \subseteq E_{\leq i-1}$, we have that Φ' is also balanced for v_h. As $\sum_{a\in E_{\leq i}} \Phi(a,j) > 0$ and $\sum_{a\in E_i^*} \phi(a,j) = 0$ for each $j \in S'$, we have that $\sum_{a\in E_{\leq i-1}} \Phi'(a,j) > 0$ for each $j \in S'$. Therefore Φ' is a witness for $\chi(E_{i-1},\phi',S') \leq \rho'$. As $E_{\leq i}$ is the disjoint union of $E_{\leq i-1}$ and E_i^*, we have that $\rho = \sum_{i\in[k]} \sum_{a\in E_{\leq i}} \Phi(a,j) = \sum_{i\in[k]} \sum_{a\in E_{\leq i-1}} \Phi'(a,j) + \sum_{j\in[k]} \sum_{a\in E_i^*} \phi(a,j) = \rho' + Y \geq \chi(E_{i-1},\phi',S') + Y$.

It remains to show that ϕ' satisfies the conditions of the recursion. As ϕ and ϕ' are both restrictions of Φ, we have that $\phi'(a,j) = \phi(a,j)$ for all $a \in E_i \cap E_{i-1}$ and all $j \in [k]$. Finally, as Φ is balanced for v_i and all arcs adjacent to v_i are contained in $E_{i-1} \cup E_i$, we have that $\phi \cup \phi'$ is balanced for v_i.

We now show that $\chi(E_i,\phi,S) \leq Y + \min_{\phi'} \chi(E_{i-1},\phi',S')$. Let $\phi' : E_{i-1} \times [k] \to [0,c]$ and be chosen such that the conditions of the recursion are satisfied and $\chi(E_{i-1},\phi',S')$ is minimized. Let $\rho' = \chi(E_{i-1},\phi',S')$ and suppose that $\rho' \neq \infty$.

Let Φ' be a witness for $\chi(E_{i-1},\phi',S') \leq \rho'$. As ϕ' agrees with ϕ, and as $E_{\leq i-1} \cap E_i = E_{i-1} \cap E_i$, we have that Φ' agrees with ϕ. Therefore the function $\Phi = \Phi' \cup \phi$ is well-defined. It is easy to see that Φ is valid, that Φ extends ϕ, that $\sum_{a\in E_{\leq i}} \Phi(a,j) > 0$ for each $j \in S$, and that $\sum_{j\in[k]} \sum_{a\in E_{\leq i}} \Phi(a,j) \cdot \omega(a) = \sum_{j\in[k]} \sum_{a\in E_{\leq i-1}} \Phi'(a,j) + \sum_{j\in[k]} \sum_{a\in E_i^*} \phi(a,j) \leq \rho' + Y$. Finally, as all arcs incident to v_h are in $E_{\leq i-1}$ for $h \leq i-1$ and Φ' is balanced for all $v_h, h \leq i-1$, we have that Φ is balanced for all $v_h, h \leq i-1$, and as all arcs incident to v_i are in $E_{i-1} \cup E_i$ and $\phi \cup \phi'$ is balanced for v_i, we have that Φ is balanced for v_i.

Note that in the above lemma, we do not need to guess the set S', as any witness Φ for $\chi(E_i,\phi,S) \leq \rho$ must have $\sum_{a \in E_{\leq i-1}} \Phi(a,j) > 0$ for each j in S' as defined in the lemma, and if a function is a witness for $\chi(E_{i-1},\phi',S'') = \rho'$ for any $S'' \supseteq S'$ then it is also a witness for $\chi(E_{i-1},\phi',S') = \rho'$.

We are now ready to prove Theorem 2.1.3.

Theorem 2.1.3 *Let (G,k) be an instance of $k[b,c]$-DWCP and suppose we are given a vertex ordering $\theta = (v_1,v_2,\ldots,v_n)$ of G with cutwidth at most p. Then, in time $O^*((c2^k)^p 2^k)$, we can solve (G,k) and find an optimal feasible solution if one exists.*

Proof Our DP algorithm calculates all values $\chi(E_i,\phi,S)$ with $\phi(\cdot,j) \leq 1$ for $j > 1$ in a bottom-up manner, that is, we only calculate values $\chi(E_i,\cdot,\cdot)$ after all values $\chi(E_j,\cdot,\cdot)$ have been calculated for $0 \leq j < i$ (we use the recursion of Lemma 2.1.10).

Each arc bag E_i of θ contains at most p arcs. For each arc a, there are $c+1$ options for $\phi(a,1)$ and 2 options for $\phi(a,j)$ for each $j > 1$, i.e., $(c+1)2^{k-1} \leq c2^k$ options per arc. Thus there are at most $(c2^k)^p$ valid choices for $\phi : E_i \times [k] \to [0,c]$. As there are 2^k choices for a set $S \subseteq [k]$, the total size of each DP table is $O((c2^k)^p 2^k)$.

Since $E_0 = \emptyset$, the only function $\phi : E_0 \times [k] \to [0,c]$ is the empty function. It is easy to see that $\chi(E_0,\phi,S) = 0$ if $S = \emptyset$, and ∞ otherwise. To speed up the application of Lemma 2.1.10 for E_i, $1 \leq i \leq n$, we form an intermediate data structure (e.g. a hash table) T from the data for bag E_{i-1}. Call two entries $\chi(E_i,\phi,S)$ and $\chi(E_{i-1},\phi',S')$ *compatible* when the conditions in Lemma 2.1.10 are met (i.e., $\chi(E_{i-1},\phi',S')$ is one of the entries included in the minimization for $\chi(E_i,\phi,S)$). Let the *signature* of entry $\chi(E_{i-1},\phi',S')$ be $(\phi'',d_1,\ldots,d_k,S')$, where ϕ'' is ϕ' restricted to arcs $E_{i-1} \cap E_i$, and where $d_j = \sum_{a \in A^+(v_i) \cap E_{i-1}} \phi'(a,j) - \sum_{a \in A^-(v_i) \cap E_{i-1}} \phi'(a,j)$ (i.e d_j is the imbalance at v_i in walk number j). Observe that whether an entry $\chi(E_{i-1},\phi',S')$ is compatible with the entry $\chi(E_i,\phi,S)$ can be determined from the signature alone, and that for each $\chi(E_i,\phi,S)$ there is at most one compatible signature. Thus, for every occuring signature $(\phi'',d_1,\ldots,d_k,S')$ we let $T(\phi'',d_1,\ldots,d_k,S')$ contain the minimum value over all entries $\chi(E_{i-1},\cdot,\cdot)$ with matching signature; this can be computed in a single loop over the entries $\chi(E_{i-1},\cdot,\cdot)$. Then, for every entry $\chi(E_i,\phi,S)$ of the new table, we look in T for the value associated with the compatible signature (and add Y to it, by Lemma 2.1.10). Note that the size of the intermediate table T is immaterial; the time taken consists of first one loop through $\chi(E_{i-1},\cdot,\cdot)$, then a single query to T for each entry in $\chi(E_i,\cdot,\cdot)$. Thus, the entries $\chi(E_i,\cdot,\cdot)$ can all be computed in total time $O^*((c2^k)^p 2^k)$. As $E_n = \emptyset$ there is only one function $\phi : E_n \times [k] \to [b,c]$. By Lemma 2.1.9, $\chi(E_n,\phi,[k])$ is the minimum total weight of a solution for $k[b,c]$-DCPP, and ∞ if there is no such solution. Thus to solve $k[b,c]$-DCPP it suffices to check the value of $\chi(E_n,\phi,[k])$.

Thus the algorithm finds the value ρ in time $O^*((c2^k)^p 2^k)$.

Using the method of Lemma 2.1.10, we can easily find an optimal solution to $k[b,c]$-DCPP For each arc bag $E_i, \phi : E_i \times [k] \to [0,c], S \subseteq [k]$, in addition to calculating the value $\chi(E_i,\phi,S) = \rho$, we also calculate a witness for $\chi(E_i,\phi,S) \leq \rho$, in the cases where $\rho \neq \infty$. Just as we can calculate the values of all $\chi(E_i,\cdot,\cdot)$ given the values of all $\chi(E_{i-1},\cdot,\cdot)$, we may construct witnesses for all $\chi(E_i,\cdot,\cdot)$ given witnesses for all $\chi(E_{i-1},\cdot,\cdot)$, using an intermediate table T as before. (Note that when $\phi : E_0 \times [k] \to [0,c]$ is the empty function, ϕ is itself a witness for $\chi(E_0,\phi,\emptyset) \leq 0$. This gives us the base case in our construction of witnesses.) Given a witness Φ for $\chi(E_n,\phi,[k]) \leq \rho$, Φ satisfies the conditions of Lemma 2.1.8. Lemma 2.1.8 shows how to construct a solution to $k[b,c]$-DCPP on G of weight at most ρ from this witness. ∎

2.1.3 Proofs of Theorems 2.1.1 and 2.1.2

Theorem 2.1.2 *The k-ADCP-*EULER *is fixed-parameter tractable.*

Proof Let D be an Euler digraph. We may assume that D has no vertex of out-degree at least k as otherwise we are done by Lemma 2.1.1. By Lemma 2.1.3, for D we can either obtain k arc-disjoint cycles or a vertex ordering θ of cutwidth at most $2g(k)$ for some function $g : \mathbb{N} \to \mathbb{N}$. Note that D is a positive instance of the k-ADCP-EULER if and only if (D,k) has a finite solution for $k[0,1]$-DWCP (as every closed walk contains a cycle). It remains to observe that the algorithm of Theorem 2.1.3 for the $k[0,1]$-DWCP is fixed-parameter when the out-degree of every vertex of D is upper-bounded by k and the cutwidth of θ is bounded by a function of k. ∎

Theorem 2.1.1 *The k-DCPP admits a fixed-parameter algorithm.*

Proof Let $G = (V,A)$ be a digraph and let T be an optimal solution of DCPP on G which we can obtain in polynomial time. Using Lemma 2.1.3, we can obtain either k arc-disjoint cycles of D or a vertex ordering of cutwidth bounded by a function of k. If we get a collection $\mathscr{C}$ of k arc-disjoint cycles in G_T, then using $\mathscr{C}$, by Lemma 2.1.5, we can solve the k-DCPP on G in (additional) polynomial time. So now assume we have a vertex ordering of G_T of bounded cutwidth. We may assume that every vertex of G_T is of out-degree at most $k-1$ (otherwise by Lemma 2.1.1, G_T has a collection of k arc-disjoint cycles). Since every vertex of G_T is of out-degree at most $k-1$, the multiplicity of G_T is at most $k-1$. Now Lemma 2.1.6 implies that there is an optimal solution W for the k-DCPP on G such that the multiplicity of G_W is at most k. Thus, we may treat the k-DCPP on G as an instance (G,k) of $k[1,k]$-DWCP. It remains to observe that the algorithm of Theorem 2.1.3 to solve the $k[1,k]$-DWCP on G will be fixed-parameter. ∎

2.1.4 k-DCPP in Planar Graphs

We would like to point out that for planar graphs, the algorithm for k-DCPP can be greatly improved. In the planar case, we don't need the procedure of running the algorithm by Grohe and Gruber as we know $f(k) = k$ by Corollary 15.3.10 in [9]. The cutwidth would be at most $2k$. The algorithm would be single exponential.

Theorem 2.1.4 *The k-DCPP in planar graphs admits a fixed-parameter algorithm which runs in time* $O^*(4^{k^2}(2k^2)^k)$.

Proof Let $G = (V,A)$ be a digraph and let T be an optimal solution of DCPP on G.

Using Lemma 2.1.3, we can obtain either k arc-disjoint cycles of T or a vertex ordering of cutwidth at most $2k$ of G. If we get a collection $\mathscr{C}$ of k arc-disjoint cycles in G_T, then using $\mathscr{C}$, by Lemma 2.1.5, we can solve the k-DCPP on G in (additional) polynomial time. So now assume we have a vertex ordering of G_T of cutwidth at most $2k$. We may assume that every vertex of G_T is of out-degree at most $k-1$ (otherwise by Lemma 2.1.1, G_T has a collection of k arc-disjoint cycles). Since every vertex of G_T is of out-degree at most $k-1$, the multiplicity of G_T is at most $k-1$. Now Lemma 2.1.6 implies that there is an optimal solution W for the k-DCPP on G such that the multiplicity of G_W is at most k. Thus, we may treat the k-DCPP on G as an instance (G,k) of $k[1,k]$-DWCP. It remains to observe that the algorithm of Theorem 2.1.3 to solve the $k[1,k]$-DWCP on G runs in time $O^*(4^{k^2}(2k^2)^k)$.
∎

2.2 Mixed k-arc CPP

A *mixed graph* is a graph that may contain both edges and arcs (i.e., directed edges). A mixed graph G is *strongly connected* if for each ordered pair x, y of vertices in G there is a path from x to y that traverses each arc in its direction.

In this section, we will study the following problem.

> MIXED CHINESE POSTMAN PROBLEM (MCPP)
> *Instance:* A strongly connected mixed graph $G = (V, E \cup A)$, with vertex set V, edge set E and arc set A; a weight function $w : E \cup A \to \mathbb{N}_0$.
> *Output:* A closed walk of G that traverses each edge and arc at least once, of minimum weight.

There is numerous literature on various algorithms and heuristics for MCPP; for informative surveys, see [102, 21, 31, 68, 74]. We call the problem the UNDIRECTED CHINESE POSTMAN PROBLEM (UCPP) when $A = \emptyset$, and the DIRECTED CHINESE POSTMAN PROBLEM (DCPP) when $E = \emptyset$. It is well-known that UCPP is polynomial-time solvable [30] and so is DCPP [14, 23, 30], but MCPP is NP-complete, even when G is planar with each vertex having total degree 3 and all edges and arcs having weight 1 [72]. It is therefore reasonable to believe that MCPP may become easier as it gets closer to UCPP or DCPP.

van Bevern *et al.* [102] considered two natural parameters for MCPP: the number of edges and the number of arcs. They showed that MCPP is fixed-parameter tractable (FPT) when parameterized with the number k of edges. That is, MCPP can be solved in time $f(k)n^{O(1)}$, where f is a function only depending on k, and n is the number of vertices in G. For background and terminology on parameterized complexity we refer the reader to the monographs [29, 35, 70]. van Bevern *et al.*'s algorithm is as follows. Replace every undirected edge uv by either the arc $\overrightarrow{uv}$ or arc $\overrightarrow{vu}$ or the pair $\overrightarrow{uv}$ and $\overrightarrow{vu}$ (all arcs have the same weight as uv) and solve the resulting DCPP. Thus, the MCPP can be solved in time $O(3^k n^3)$, where n is the number of the number of vertices in G.

We describe a faster algorithm here. We will build circulation instances from G with varying lower bounds of the flow for some arcs. Replace every undirected edge uv by the arc pair $\overrightarrow{uv}$ and $\overrightarrow{vu}$. Now construct a network N from the resulting digraph D as follows: the cost of every arc of G is the same as its weight, and the cost of every arc $\overrightarrow{xy}$ in D, which is not in G, is the weight the undirected edge xy of G. The lower bound of every arc of G is 1, and for each pair $\overrightarrow{uv}$ and $\overrightarrow{vu}$ of arcs that replaced an undirected edge uv, we assign lower bound 0 to one of the arcs and 1 to the other. All upper bounds are ∞. Find a minimum-cost circulation (i.e., a flow of value 0) in N. This will correspond to a closed walk in D in which all arcs of G are traversed at least once and at least one of the arcs $\overrightarrow{uv}$ and $\overrightarrow{vu}$ corresponding to an

undirected edge uv of G is traversed at least once (the arc whose lower bound is 1 in N). As there are 2^k ways to assign lower bounds to the pairs of arcs in N, we obtain a running time of $O(2^k n^3)$.

van Bevern *et al.* [102] and Sorge [90] left it as an open question whether the MCPP is fixed-parameter tractable when parameterized with the number of arcs. This is the parameterization we consider in this paper.

> k-ARC CHINESE POSTMAN PROBLEM (k-ARC CPP)
> *Instance:* A strongly connected weighted mixed graph $G = (V, E \cup A)$, with vertex set V, edge set E and arc set A; a weight function $w : E \cup A \to \mathbb{N}_0$.
> *Parameter:* $k = |A|$.
> *Output:* A closed walk of G that traverses each edge and arc at least once, of minimum weight.

This parameterized problem is of practical interest, for example, if we view the mixed graph as a network of streets in a city: while edges represent two-way streets, arcs are for one-way streets. Many cities have a relatively small number of one-way streets and so the number of arcs appears to be a good parameter for optimizing, say, police patrol in such cities [102].

We will assume for convenience that the input G of k-ARC CPP is a *simple* graph, i.e. there is at most one edge or one arc (but not both) between any pair of vertices. The multigraph version of the problem may be reduced to the simple graph version by subdividing arcs and edges. As the number of arcs and edges is at most doubled by this reduction, this does not affect the parameterized complexity of the problem.

We will show that k-ARC CPP is fixed-parameter tractable. Our proof is significantly more complicated than the ones described above for the MCPP parameterized with the number of edges. For that problem, as we saw, we can replace the undirected edges with arcs. However a similar approach for MCPP parameterized with the number of arcs (replacing arcs with edges) does not work. Instead, in FPT time, we reduce the problem to the BALANCED CHINESE POSTMAN PROBLEM (BCPP), in which there are no arcs, but instead a demand function on the imbalance of the vertices is introduced (the parameter for the BCPP is based on the values of the demand function). This reduction is only the first step of our proof, as unfortunately the BCPP is still NP-hard, unlike the DCPP.

The BCPP turns out to be polynomial time solvable as long as a certain connectivity property holds. Solving the problem in general requires making some guesses on the edges in certain small cuts in the graph. To keep the running time fixed-parameter, we require a structure on the graph that allows us to only consider a few edges from small cuts at a time.

To achieve this, we make use of a recent result of Marx, O'Sullivan and Razgon [66] on the treewidth of torso graphs with respect to small separators.

Marx, O'Sullivan and Razgon [66] use the following notion of a graph torso. Let $G = (V,E)$ be a graph and $S \subseteq V$. The graph torso(G,S) has vertex set S and vertices $a,b \in S$ are connected by an edge ab if $ab \in E$ or there is a path in G connecting a and b whose internal vertices are not in S.

Marx *et al.* [66] show that for a number of graph separation problems, it is possible to derive a graph closely related to a torso graph, which has the same separators as the original input graph. The separation problem can then be solved on this new graph, which has bounded treewidth. By contrast, we use the torso graph as a tool to construct a tree decomposition of the original graph, which does not have bounded width, but has enough other structural restrictions to make a dynamic programming algorithm possible. So, our application of Marx *et al.*'s result is quite different from its use in [66], and we believe it may be used for designing fixed-parameter algorithms for other problems on graphs. Note that Marx *et al.* are interested in small separators (i.e. sets of vertices whose removal disconnects a graph), whereas we are interested in small cuts (sets of edges whose removal disconnects a graph). We therefore prove an analog of Marx *et al.*'s result for cuts.

The rest of the section is organized as follows. The next subsection contains further terminology and notation. In subsection 2.2.2, we reduce k-ARC CPP to BALANCED CHINESE POSTMAN PROBLEM (BCPP). In subsection 2.2.3, we introduce and study two key notions that we use to solve BCPP: t-roads, which witness a connectivity property of the graph that makes the BCPP easy to solve; and small t-cuts, which witness the fact that a t-road does not exist. In subsection 2.2.4, we investigate a special tree decomposition of the input graph of BCPP. This decomposition is used in a dynamic programming algorithm given in Section 2.2.5.

Note that our algorithm for k-arc CPP can be easily extend to Windy Postman Problem, just guess for each windy edge the directions we are going to traverse through it. There would be at most 3^k guesses, after which, it remains to solve BCPP. As far as we know, this is the first FPT result for Windy Postman problem. There are not many positive results for windy postman problem, except the fact that it is polynomial time solvable in Euler graphs [106].

WINDY POSTMAN PROBLEM (WPP)
Instance: A connected, undirected graph $G = (V,E)$, with two nonnegative cost value $c(uv), c(vu)$ for every edge $uv \in E$.
Parameter: $k = |\{uv : c(uv) \neq c(vu)\}|$.
Output: An Eulerian multigraph of minimum weight that contains at least one copy of every edge in E.

2.2.1 Further Terminology and Notation

To avoid confusion, we denote an edge between two vertices u, v as uv, and an arc from u to v as $\overrightarrow{uv}$.

Although we will shortly reduce the the k-ARC CPP to a problem on undirected graphs, we will still be interested in directed graphs as a way of expressing solutions. For example, a walk which is a solution to an instance of the k-ARC CPP can be represented by a directed multigraph, with one copy of an arc uv for each time the walk passes from u to v. This motivates the following definitions.

For a mixed multigraph G, $\mu_G(\overrightarrow{uv})$ denotes the number of arcs of the form $\overrightarrow{uv}$ in G, and $\mu_G(uv)$ denotes the number of edges of the form uv. For a mixed multigraph G, let D be a directed multigraph derived from G by replacing each arc $\overrightarrow{uv}$ of G with multiple copies of $\overrightarrow{uv}$ (at least one), and replacing each edge uv in G with multiple copies of the arcs $\overrightarrow{uv}$ and $\overrightarrow{vu}$ (such that there is at least one copy of $\overrightarrow{uv}$ or at least one copy of $\overrightarrow{vu}$). Then we say D is a *multi-orientation* of G. If D is a multi-orientation of G and $\mu_D(\overrightarrow{uv}) + \mu_D(\overrightarrow{vu}) = \mu_G(\overrightarrow{uv}) + \mu_G(uv) + \mu_G(\overrightarrow{vu})$ for each $u, v \in V$ (i.e. D is derived from G by keeping every arc of G and replacing every edge of G with a single arc), we say D is an *orientation* of G. If D is an orientation of G and G is undirected, we say that G is the *undirected version* of D. The *underlying graph* G of an undirected multigraph H can be obtained from H by deleting all but one edge among all edges between u and v for every pair u, v of vertices of H.

For a simple weighted graph G and a multi-orientation D of G, the *weight* of D is the sum of the weights of all its arcs, where the weight of an arc in D is the weight of the corresponding edge or arc in G.

For a directed multigraph $D = (V,A)$ and $v \in V$, $d_D^+(v)$ and $d_D^-(v)$ denote the out-degree and in-degree of v in D, respectively. Let $t : V \to \mathbb{Z}$ be a function and $V_t^+ = \{u \in V, t(u) > 0\}, V_t^- = \{u \in V, t(u) < 0\}$. We say that a vertex u in D is *t-balanced* if $d_D^+(u) - d_D^-(u) = t(u)$. We say that D is *t-balanced* if every vertex is t-balanced. Note that if D is t-balanced then $\sum_{v \in V} t(v) = 0$. We say that a vertex u in D is *balanced* if $d_D^+(u) = d_D^-(u)$, and we say that D is *balanced* if every vertex is balanced. Note that D is balanced if and only if it is t_0-balanced, where $t_0(v) = 0$ for all $v \in 0$.

In directed multigraphs, all walks (in particular, paths and cycles) that we consider are directed. A directed multigraph D is *Eulerian* if there is a closed walk of D traversing every arc exactly once. It is well-known that a directed multigraph D is Eulerian if and only if D is balanced and the undirected version of D is connected [9].

For an undirected graph $G=(V,E)$, and two vertex sets $X,Y \subseteq V(G)$, an (X,Y)-*separator* ((X,Y)-*cut*, respectively) is a set $S \subseteq V \setminus (X \cup Y)$ (a set $S \subseteq E$, respectively) such that there is no path between vertices in X and Y in graph $G-S$. When $X=\{x\}$ and $Y=\{y\}$, we speak of (x,y)-separators and (x,y)-cuts.

Observe that the following is an equivalent formulation of the k-ARC CPP.

k-ARC CHINESE POSTMAN PROBLEM (k-ARC CPP)
Instance: A strongly connected mixed graph $G=(V,E \cup A)$, with vertex set V, edge set E and arc set A; weight function $w : E \cup A \to \mathbb{N}_0$.
Parameter: $k=\|A\|$.
Output: A directed multigraph D of minimum weight such that D is a multi-orientation of G and D is Eulerian.

2.2.2 Reduction to Balanced CPP

Our first step is to reduce k-ARC CPP to a problem on a graph without arcs. Essentially, given a graph $G=(V,E \cup A)$ we will "guess" the number of times each arc in A is traversed in an optimal solution. This then leaves us with a problem on $G'=(V,E)$. Rather than trying to find an Eulerian multi-orientation of G, we now try to find a multi-orientation of G' in which the imbalance between the in- and out-degrees of each vertex depends on the guesses for the arcs in A incident with that vertex.

More formally, we will provide a Turing reduction to the following problem:

BALANCED CHINESE POSTMAN PROBLEM (BCPP)
Instance: An undirected graph $G=(V,E)$; a weight function $w : E \to \mathbb{N}_0$; a demand function $t : V \to \mathbb{Z}$ such that $\sum_{v \in V} t(v) = 0$. *Parameter:* $p = \sum_{v \in V_t^+} t(v)$.
Output: A minimum weight t-balanced multi-orientation D of G.

Henceforth, any demand function $t : V \to \mathbb{Z}$ will be such that $\sum_{v \in V} t(v) = 0$.

Observe that when $t(v)=0$ for all $v \in V$, BCPP is equivalent to UCPP. BCPP was studied by Zaragoza Martínez [110] who proved that the problem is NP-hard. We will reduce k-ARC CPP to BCPP by guessing the number of times each arc is traversed. In order to ensure a fixed-parameter algorithm, we need a bound (in terms of $|A|$) on the number of guesses. We will do this by bounding the total number of times any arc can be traversed in an optimal solution.

Lemma 2.2.1 *Let $G = (V, A \cup E)$ be a mixed graph, and let $k = |A|$. Then for any optimal solution D to k-ARC CPP on G with minimal number of arcs, we have that $\sum_{\overrightarrow{uv} \in A} \mu_D(\overrightarrow{uv}) \leq k^2/2 + 2k$.*

Proof Let $A = A_1 \cup A_2$ where $A_1 = \{\overrightarrow{uv} : \overrightarrow{uv} \in A \text{ and } \mu_D(\overrightarrow{uv}) \geq 3\}$ and $A_2 = A \setminus A_1$. Let $A_3 = \{\overrightarrow{uv} \in A(D) : uv \in E(G)\}$. Let $|A_1| = p$ and $|A_2| = k - p = q$.

Consider an arc $\overrightarrow{uv} \in A$. Since D is balanced, it follows that D has $\mu_D(\overrightarrow{uv})$ arc-disjoint directed cycles and each of these cycle contains exactly one copy of $\overrightarrow{uv}$. We claim that each such cycle must contain at least one copy of an arc in A_2. Indeed, otherwise, there is a cycle C containing $\overrightarrow{uv}$ that does not contain any arc in A_2, which means that C consists of arcs in A_1 and arcs in A_3. We may construct a directed multigraph D' as follows: Remove from D two copies of each arc in A_1 that appears in C, and reverse the arcs in $A_3 \cap C$. Observe that D' is Eulerian and is also a multi-orientation of G, and so D' is a solution with smaller weight than D or an optimal solution with fewer arcs than D, contradicting the minimality of D.

So each of the $\mu_D(\overrightarrow{uv})$ cycles contains at least one copy of an arc in A_2. Observe that D has at most $2q$ copies of arcs in A_2, and so $\mu_D(\overrightarrow{uv}) \leq 2q$. Thus, we have $\sum_{\overrightarrow{uv} \in A} \mu_D(\overrightarrow{uv}) = \sum_{\overrightarrow{uv} \in A_1} \mu_D(\overrightarrow{uv}) + \sum_{\overrightarrow{uv} \in A_2} \mu_D(\overrightarrow{uv}) \leq p \cdot 2q + 2q \leq 2 \cdot (\frac{p+q}{2})^2 + 2k = k^2/2 + 2k$. ∎

Now we may prove the following:

Lemma 2.2.2 *Suppose that there exists an algorithm which finds the optimal solution to an instance of* BCPP *on (G', w', t') with parameter p in time $f(p)|V(G')|^{O(1)}$. Then there exists an algorithm which finds the optimal solution to an instance of k-ARC CPP on $(G = (V, A \cup E), w)$ with parameter k, which runs in time $\binom{\lfloor k^2/2+2k \rfloor}{k} \cdot f(\lfloor k^2/2 + 2k \rfloor) \cdot |V|^{O(1)}$.*

Thus, if BCPP *is FPT then so is k-ARC CPP.*

Proof Let $(G = (V, A \cup E), w)$ be an instance of k-ARC CPP, and let $k = |A|$. Let $\kappa = \lfloor k^2/2 + 2k \rfloor$. By Lemma 2.2.1, $\sum_{\overrightarrow{uv} \in A} \mu_D(\overrightarrow{uv}) \leq \kappa$ for any optimal solution D to k-ARC CPP on (G, w) with minimal number of arcs.

Let $G' = (V, E)$ and let w' be w restricted to E. Given a function $\phi : A \to [\kappa]$ such that $\sum_{\overrightarrow{uv} \in A} \phi(\overrightarrow{uv}) \leq \kappa$, let $t_\phi : V \to [-\kappa, \kappa]$ be the function such that $t_\phi(v) = \sum_{\overrightarrow{uv} \in A} \phi(\overrightarrow{uv}) - \sum_{\overrightarrow{vu} \in A} \phi(\overrightarrow{vu})$ for all $v \in V$. Observe that $\sum_{v \in V^+_{t_\phi}} t_\phi(v) \leq \sum_{\overrightarrow{uv} \in A} \phi(\overrightarrow{uv}) \leq \kappa$, and thus BCPP on (G', w', t_ϕ) has parameter $p_\phi \leq \kappa$.

Observe that for any given solution D_ϕ to BCPP on (G', w', t_ϕ), D_ϕ is t_ϕ-balanced, thus $d^+_{D_\phi}(u) - d^-_{D_\phi}(u) = t_\phi(u)$. If we add $\phi(\overrightarrow{uv})$ copies of each arc $\overrightarrow{uv} \in A$ to D_ϕ, and denote the resulting graph with D, then graph D is balanced. Indeed for any vertex $v \in V(G)$, $d^+_D(v) = d^+_{D_\phi}(v) + \sum_{\overrightarrow{vu} \in A} \phi(\overrightarrow{vu})$, $d^-_D(v) = d^-_{D_\phi}(v) + \sum_{\overrightarrow{uv} \in A} \phi(\overrightarrow{uv})$, thus $d^+_D(v) - d^-_D(v) = d^+_{D_\phi}(v) - d^-_{D_\phi}(v) + \sum_{\overrightarrow{vu} \in A} \phi(\overrightarrow{vu}) - \sum_{\overrightarrow{uv} \in A} \phi(\overrightarrow{uv}) = t_\phi(v) - t_\phi(v) = 0$. So D is a connected

balanced graph (and thus Eulerian) which is also a multi-orientation of G, and thus is a solution to k-ARC CPP on (G,w) with weight $w'(D_\phi)+\sum_{\overrightarrow{uv}\in A}\phi(\overrightarrow{uv})w(\overrightarrow{uv})$.

Furthermore for any solution D to k-ARC CPP on (G,w), we know that D is balanced, so for any vertex $v\in V(G)$, $d_D^+(v)=d_D^-(v)$. Let $\phi(\overrightarrow{uv})=\mu_D(\overrightarrow{uv})$ for each $\overrightarrow{uv}\in A$ and let D_ϕ be D restricted to E. For any vertex $v\in V(G)$, we have $d^+_{D_\phi}(v)=d^+_D(v)-\sum_{\overrightarrow{vu}\in A}\phi(\overrightarrow{vu})$, $d^-_{D_\phi}(v)=d^-_D(v)-\sum_{\overrightarrow{uv}\in A}\phi(\overrightarrow{uv})$, therefore, $d^+_{D_\phi}(v)-d^-_{D_\phi}(v)=\sum_{\overrightarrow{uv}\in A}\phi(\overrightarrow{uv})-\sum_{\overrightarrow{vu}\in A}\phi(\overrightarrow{vu})=t_\phi(v)$. So D_ϕ is a t_ϕ-balanced multi-orientation of G' and thus a solution to BCPP on (G',w',t_ϕ) and D has weight $w'(D_\phi)+\sum_{\overrightarrow{uv}\in A}\phi(\overrightarrow{uv})w(\overrightarrow{uv})$.

There are at most $\binom{q}{k}$ ways of choosing positive integers $x_1,\ldots,x_k$ such that $\sum_{i\in[k]}x_i\le q$. Indeed, for each $i\in[k]$ let $y_i=\sum_{j=1}^{i}x_j$. Then $y_i<y_j$ for $i<j$ and $y_i\in[q]$ for all i, and for any such choice of $y_1,\ldots,y_k$ there is corresponding choice of $x_1,\ldots,x_k$ satisfying $\sum_{i=1}^{k}x_i\le q$. Therefore the number of valid choices for $x_1,\ldots,x_k$ is the number of ways of choosing $y_1,\ldots,y_k$, which is the number of ways of choosing k elements from a set of q elements.

Therefore there are at most $\binom{\kappa}{k}$ choices for a function $\phi:A\to[\kappa]$ such that $\sum_{\overrightarrow{uv}\in A}\phi(\overrightarrow{uv})\le\kappa$. Each choice leads to an instance of BCPP with parameter at most κ. Therefore in time $\binom{\kappa}{k}f(\kappa)\cdot|V|^{O(1)}$ we can find, for every valid choice of ϕ, the optimal solution D_ϕ to BCPP on (G',w',t_ϕ).

It then remains to choose the function ϕ that minimizes $w'(D_\phi)+\sum_{\overrightarrow{uv}\in A}\phi(\overrightarrow{uv})w(\overrightarrow{uv})$, and return the graph D_ϕ together with $\phi(\overrightarrow{uv})$ copies of each arc $\overrightarrow{uv}\in A$. ∎

Due to Lemma 2.2.2, we may now focus on BCPP.

2.2.3 Expressing Connectivity: t-roads and t-cuts

Although we will not need the result until later, now is a good time to prove a bound for BCPP somewhat similar to that in Lemma 2.2.1.

Lemma 2.2.3 *Let (G,w,t) be an instance of* BCPP*, with $p=\sum_{v\in V_t^+}t(v)$. Then for any optimal solution D to* BCPP *on (G,w,t) with minimal number of arcs, we have that $\mu_D(\overrightarrow{uv})+\mu_D(\overrightarrow{vu})\le\max\{p,2\}$ for each edge uv in G.*

Proof Suppose that $\mu_D(\overrightarrow{uv})+\mu_D(\overrightarrow{vu})>\max\{p,2\}$ for some edge uv in G. Observe that if $\mu_D(\overrightarrow{uv})\ge1$ and $\mu_D(\overrightarrow{vu})\ge1$, then by removing one copy of $\overrightarrow{uv}$ and one copy of $\overrightarrow{vu}$, we obtain a solution to BCPP on (G,w,t) with weight at most that of D but with fewer arcs. (Note that $\mu_D(\overrightarrow{uv})-1+\mu_D(\overrightarrow{vu})-1>2-2=0$, and so we still have a solution). Therefore, we may assume that $\mu_D(\overrightarrow{uv})>\max\{p,2\}$ and $\mu_D(\overrightarrow{vu})=0$.

We now show that there must exist a cycle in D containing a copy of $\overrightarrow{uv}$.

Modify D by adding a new vertex x, with $t(v)$ arcs from x to v for each $v \in V_t^+$, and $-t(v)$ arcs from v to x for each $v \in V_t^-$. Let D^* be the resulting directed graph. Then observe that D^* is balanced, and therefore D^* has $\mu_D(\overrightarrow{uv})$ arc-disjoint cycles, each containing exactly one copy of $\overrightarrow{uv}$. At most p of these cycles can pass through x. Therefore there is at least one cycle containing $\overrightarrow{uv}$ which is a cycle in D.

So now let $v = v_1, v_2, \ldots, v_l = u$ be a sequence of vertices such that $\mu_D(\overrightarrow{v_i v_{i+1}}) \geq 1$ for each $i \in [l-1]$. Replace one copy of each arc $\overrightarrow{v_i v_{i+1}}$ with a copy of $\overrightarrow{v_{i+1} v_i}$ and remove 2 copies of $\overrightarrow{uv}$. Observe that the resulting graph covers every edge of G, and the imbalance of each vertex is the same as in D. Therefore, we have a solution to BCPP on (G,w,t) with weight at most that of D but with fewer arcs. This contradiction proves the lemma. ∎

In order to solve the BCPP on a graph G, we first add copies of edges to G to produce a multigraph H, and then find an orientation of H which is a solution to the BCPP on G. Thus, H is the undirected version of a solution to the BCPP on G. The lemma below gives a connectivity condition which must be satisfied by any undirected version of a solution. Furthermore, any multigraph that satisfies this condition has an orientation which is a solution to BCPP, and such a solution can be found in polynomial time. We will then be able to solve the BCPP on G by searching for the minimum weight graph H that satisfies this condition.

Definition 2.2.1 *Let $H = (V,E)$ be an undirected multigraph and t a demand function $V \to \mathbb{Z}$. A t-*road *is a directed multigraph T with vertex set V such that for each vertex $v \in V$, $d_T^+(v) - d_T^-(v) = t(v)$. We say H* has a *t-*road *T if there is a subgraph H' of H such that T is an orientation of H'.*

For an instance (G,w,t) of the BCPP with parameter p, it may be useful to think of a t-road as a set of p arc-disjoint paths from vertices in V_t^+ to vertices in V_t^-, although a t-road does not necessarily have to have such a simple structure. Indeed, a t-road may also contain several closed walks. In particular, we note that any solution to the BCPP on (G,w,t) is itself a t-road.

The following lemma and corollary show the relevance of t-roads to the BCPP. Formally an input of BCPP is a simple graph, but to show Corollary 2.2.1 we will abuse this formality and allow multigraphs.

Lemma 2.2.4 *Let H be an undirected multigraph and let (H,w,t) be an instance of the* BCPP. *Then (H,w,t) has a solution which is an orientation of H (which is necessarily an optimal solution) if and only if H has a t-road and for every vertex v of H, $d_H(v) - t(v)$ is even. Furthermore, such a solution can be found in polynomial time.*

Proof Suppose first that (H,w,t) has a solution of weight $w(H)$. Then there is a directed multigraph D with vertex set $V(H)$ such that D is an orientation of H, and $d_D^+(v) - d_D^-(v) = t(v)$ for every vertex $v \in V(H)$. Thus, D itself is a t-road which is an orientation of a subgraph of H, and so H has a t-road. Furthermore, for every vertex v of H, $d_H(v) - t(v) = d_D^+(v) + d_D^-(v) - t(v) = d_D^+(v) - d_D^-(v) - t(v) + 2d_D^-(v) = 2d_D^-(v)$, which is even.

Conversely, suppose that H has a t-road and for every vertex v of H, $d_H(v) - t(v)$ is even. Let T be a t-road in H. Delete the edges corresponding to T from H, and observe that in the remaining graph every vertex v has degree $d_H(v) - d_T^+(v) - d_T^-(v) = d_H(v) - d_T^+(v) + d_T^-(v) - 2d_T^-(v) = d_H(v) - t(v) - 2d_T^-(v)$, which is even. Thus in this remaining graph every vertex is of even degree, and so we may decompose the remaining edges into cycles. Orient each of these cycles arbitrarily, and finally add the arcs of T. Let D be the resulting digraph. Then for each vertex $v \in V(H)$, $d_D^+(v) - d_D^-(v) = 0 + d_T^+(v) - d_T^-(v) = t(v)$. Thus D is t-balanced and is an orientation of H, as required. ∎

By letting H be the undirected version of an optimal solution to an instance (G,w,t), we get the following corollary.

Corollary 2.2.1 *Given an instance (G,w,t) of the* BCPP*, let H be an undirected multigraph of minimum weight, such that the underlying graph of H is G, H has a t-road, and $d_H(v) - t(v)$ is even for every vertex v. Then there exists an optimal solution to (G,w,t) which is an orientation of H, which can be found in polynomial time.*

Suppose that G has a t-road. Then by the above corollary, it is enough to find a minimum weight multigraph H with underlying graph G, such that $d_H(v) - t(v)$ is even for every vertex v. This can be done in polynomial time as follows.

Given a graph $G = (V,E)$ and set $X \subseteq V$ of vertices, an *X-join* is a set $J \subseteq E$ such that $|J(v)|$ is odd if and only if $v \in X$, where $J(v) \subseteq J$ is the set of edges incident to v. Let X be the set of vertices such that $d_H(v) - t(v)$ is odd. Note that if J is an X-join of minimum weight, the mutigraph $H = (V, E \cup J)$ is a minimum weight multigraph with underlying graph G, such that $d_H(v) - t(v)$ is even for every vertex v.

Thus, to solve the BCPP on (G,w,t), where G has a t-road, it is enough to find a minimum weight X-join. This problem is known as the MINIMUM WEIGHT X-JOIN PROBLEM (traditionally, it is called the MINIMUM WEIGHT T-JOIN PROBLEM, but we use T for t-roads) and can be solved in polynomial time:

Lemma 2.2.5 *[30] The* MINIMUM WEIGHT X-JOIN PROBLEM *can be solved in time $O(n^3)$.*

Let us briefly recall the proof of Lemma 2.2.5. Create a graph with vertex set X. For any two vertices $u,v \in X$, create an edge uv of weight equal to the minimum weight of a path

between u and v in G. Find the minimum weight perfect matching in this graph. Then the weight of this matching is the weight of an X-join, and an X-join can be found by taking the paths corresponding to edges in the matching.

The above remark shows that if G has a t-road, then we can solve the BCPP in polynomial time. In general, G may not have a t-road. However, given a solution D to the BCPP on (G,w,t), the undirected version of D must have a t-road (indeed, D itself is a t-road). Therefore if we can correctly guess the part of a solution corresponding to a t-road, and amend G using this partial solution, the rest of the problem becomes easy. The following definition and lemmas allow us to restrict such a guess to the places where there are small cuts that prevent a t-road from existing.

Definition 2.2.2 *Let $H = (V,E(H))$ be an undirected multigraph and $t : V \to \mathbb{Z}$ a demand function such that $\sum_{v\in V} t(v) = 0$. Let $p = \sum_{v\in V_t^+} t(v)$. Then a* small t-cut *of H is a minimal (V_t^+, V_t^-)-cut F such that $|F| < p$.*

A t-road in H, if one exists, can be found in polynomial time by computing a flow of value p from a to b in the unit capacity network N with underlying multigraph H^*, where H^* is the multigraph derived from H by creating two new vertices a,b, with $t(v)$ edges between a and v for each $v \in V_t^+$, and $-t(v)$ edges between b and v for each $v \in V_t^-$. The next lemma follows from the well-known max-flow-min-cut theorem for N.

Lemma 2.2.6 *An undirected multigraph H has a t-road if and only if H does not have a small t-cut.*

The next lemma shows that if we want to decide where to duplicate edges to get a t-road, we can restrict our attention to the edges in small t-cuts.

Definition 2.2.3 *Let $G = (V,E)$ be an undirected graph and $t : V \to \mathbb{Z}$ a demand function. Let $F(G)$ be the union of all small t-cuts in G. Then a directed multigraph T is* well-(G,t)-behaved *if $\mu_T(\overrightarrow{uv}) = 0$ for all $uv \notin E$ and $\mu_T(\overrightarrow{uv}) + \mu_T(\overrightarrow{vu}) \leq 1$ for all $uv \in E \setminus F(G)$.*

Lemma 2.2.7 *Let D be an optimal solution to* BCPP *on $(G = (V(G),E(G)),w,t)$, and let H be the underlying graph of D. Then H has a well-(G,t)-behaved t-road.*

Proof Let $F(G) \subseteq E(G)$ be the union of all small t-cuts in G. Let J be the undirected multigraph derived from H by removing all but one copy of every edge in $E(G) \setminus F(G)$. Note that $V(G) = V(H) = V(J)$, $E(G) \subseteq E(J) \subseteq E(H)$, moreover, H and J have the same weight function w as G. Observe that every (well-(G,t)-behaved) t-road in J is also a (well-(G,t)-behaved) t-road in H and every t-road in J is well-(G,t)-behaved. So, it is sufficient to show that J has a t-road.

Note that if J does not have a t-road, then by Lemma 2.2.6, J has a small t-cut. Note also that by construction, D is a t-road, thus H has a t-road and therefore does not have a small t-cut. Consider a small t-cut S in J and suppose that every edge in S is a copy of an edge in $F(G)$. As J contains the same number of copies of each edge in $F(G)$ as H does, it follows that any path from $u \in V_t^+$ to $v \in V_t^-$ in $H \setminus S$ is also a path from u to v in $J \setminus S$. But as H does not have a small t-cut, such a path must exist, contradicting the assumption that S is a small t-cut in J. Therefore every small t-cut in J contains a copy of an edge not in $F(G)$. If J has a small t-cut, then as G is a subgraph of J, every small t-cut in J is also a small t-cut in G, it follows that there is a small t-cut in G containing edges not in F. This is a contradiction by definition of F. Therefore we may conclude that J does not have a small t-cut, and so J has a t-road, as required. ∎

If $|F(G)|$, the number of edges of G in small t-cuts, is bounded by a function on p then, using Lemma 2.2.3 and Lemma 2.2.7 we can solve BCPP in FPT time by guessing the multiplicities of each edge in F for an optimal solution D. Unfortunately, $|F(G)|$ may be larger than any function of p in general. It is also possible to solve the problem on graphs of bounded treewidth using dynamic programming techniques, but in general the treewidth may be unbounded. In Section 2.2.4 we give a tree decomposition of G in which the number of edges from $F(G)$ in each bag is bounded by a function of p. This allows us to combine both techniques. In Section 2.2.5 we give a dynamic programming algorithm utilizing Lemma 2.2.7 that runs in FPT time.

2.2.4 Tree Decomposition

Definition 2.2.4 *Given an undirected graph $G = (V,E)$, a* tree decomposition *of G is a pair $(\mathscr{T},\beta)$, where $\mathscr{T}$ is a tree and $\beta : V(\mathscr{T}) \to 2^V$ such that*

1. $\bigcup_{x \in V(\mathscr{T})} \beta(x) = V$;
2. *for each edge $uv \in E$, there exists a node $x \in V(\mathscr{T})$ such that $u,v \in \beta(x)$; and*
3. *for each $v \in V$, the set $\beta^{-1}(v)$ of nodes form a connected subgraph in $\mathscr{T}$.*

The width *of $(\mathscr{T},\beta)$ is* $\max_{x \in V(\mathscr{T})}(|\beta(x)| - 1)$. *The* treewidth *of G (denoted $tw(G)$) is the minimum width of all tree decompositions of G.*

In this section, we provide a tree decomposition of G which we will use for our dynamic programming algorithm. The tree decomposition does not have bounded treewidth (i.e. the bags do not have bounded size), but the intersection between bags is small, and each bag has

a bounded number of vertices from small t-cuts. This will turn out to be enough to develop a fixed-parameter algorithm, as in some sense the hardness of BCPP comes from the small t-cuts.

Our tree decomposition is based on a result by Marx, O'Sullivan and Razgon [66], in which they show that the minimal small separators of a graph "live in a part of the graph that has bounded treewidth"[66].

Definition 2.2.5 *Let G be a graph and $C \subseteq V(G)$. The graph torso(G,C) has vertex set C and vertices $a,b \in C$ are connected by an edge if $ab \in E(G)$ or there is a path P in G connecting a and b whose internal vertices are not in C.*

Lemma 2.2.8 *[66, Lemma 2.11] Let a,b be vertices of a graph $G = (V,E)$ and let l be the minimum size of an (a,b)-separator. For some $e \geq 0$, let S be the union of all minimal (a,b)-separators of size at most $l+e$. Then there is an $f(l,e) \cdot (|E|+|V|)$ time algorithm that returns a set $S' \supseteq S$ disjoint from $\{a,b\}$ such that $tw(torso(G,S')) \leq g(l,e)$, for some functions f and g depending only on l and e.*

Marx et al.'s result concerns small separators (i.e. sets of vertices whose removal disconnects a graph), whereas we are interested in small cuts (sets of edges whose removal disconnects a graph). For this reason, we prove the "edge version" of Marx et al.'s result, which follows directly from their version.

Lemma 2.2.9 *Let a,b be vertices of a graph $G = (V,E)$ and let l be the minimum size of an (a,b)-cut. For some $e \geq 0$, let D be the union of all minimal (a,b)-cuts of size at most $l+e$, and let $C = V(D) \setminus \{a,b\}$. Then there is an $f(l,e) \cdot (|E|+|V|)$ time algorithm that returns a set $C' \supseteq C$ disjoint from $\{a,b\}$ such that $tw(torso(G,C')) \leq g(l,e)$, for some functions f and g depending only on l and e.*

Proof We may assume that $ab \notin E$, as otherwise all minimal (a,b)-cuts must contain ab and deleting ab from G will not change the set C as it is disjoint from $\{a,b\}$.

The main idea is to augment G to produce a graph G^* such that every vertex in C is part of a minimal (a,b)-separator in G^*. We then apply Lemma 2.2.8 to get a set $S' \supseteq C$ and tree decomposition of torso(G^*,S') of bounded width, and then use this to produce a set $C' \supseteq C$ and tree decomposition of torso(G,C') of bounded width.

We first produce the graph G^* by subdividing each edge f in G with a new vertex v_f. Let S be the union of all minimal (a,b)-separators in G^* of size at most $l+e$. Let l' be the minimum size of an (a,b)-separator in G^* (note that l' may be different from l).

Observe that for any minimal (a,b)-cut F of size at most $l+e$ in G, the set $\{v_f : f \in F\}$ is a minimal (a,b)-separator in G^*. This implies that $l' \leq l$. Furthermore, given any edge

$f' = uv$ such that $f' \in F$, assuming $u \notin \{a,b\}$, the set $(\{v_f : f \in F\} \setminus \{v_{f'}\}) \cup \{u\}$ is an (a,b)-separator in G^* and $\{v_f : f \in F\} \setminus \{v_{f'}\}$ is not. So, $X \cup \{u\}$ is a minimal (a,b)-separator in G^* for some $X \subseteq \{v_f : f \in F\} \setminus \{v_{f'}\}$. Therefore, u is in a minimal (a,b)-separator in G^* of size less than $l+e$. A similar argument holds for v. It follows that $\{u,v\} \setminus \{a,b\} \subseteq S$ for any edge $uv \in D$, and so $C \subseteq S$.

Let $e' = (l-l')+e$, so we have $l'+e' = l+e$. Now apply Lemma 2.2.8 to get a set S' disjoint from $\{a,b\}$ such that $S \subseteq S'$, and a tree decomposition $(\mathscr{T},\beta')$ of torso(G^*,S') with treewidth at most $g(l',e')$. As $l' \leq l$ and $e' \leq l+e$, this treewidth is bounded by a function depending only on l and e.

Define a function $h : S' \to V(G)$ as follows. For each edge $f \in E(G)$ such that $v_f \in S'$, if a or b is an endpoint of f, set $h(v_f)$ to be the other endpoint, and otherwise let $h(v_f)$ be an arbitrary endpoint of f. For every other $v \in S$, let $h(v) = v$. Now let $C' = \{h(v) : v \in S'\}$. Observe that $C \subseteq S' \cap V(G) \subseteq C'$.

We produce a tree decomposition of torso(G,C') as follows. Given the tree decomposition $(\mathscr{T},\beta')$ of torso(G^*,S'), define $\beta : V(\mathscr{T}) \to C'$ by $\beta(x) = h(\beta'(x)) = \{h(v) : v \in \beta'(x)\}$. We now show that $(\mathscr{T},\beta)$ is indeed a tree decomposition of torso(G,C').

It follows from construction that $\bigcup_{x \in V(\mathscr{T})} \beta(x) = C' = V(\text{torso}(G,C'))$.

Now consider an edge uw in torso(G,C'). We will show that there is an edge st in torso(G^*,S') with $h(s) = u, h(t) = w$. It follows that $s,t \in \beta'(x)$ for some node $x \in V(\mathscr{T})$, and consequently $u,w \in \beta(x)$ for the same node x. This satisfies the second condition of the tree decomposition.

As u,w are adjacent in torso(G,C'), there must be a path between them which has no internal vertices in C'. By subdividing each edge f in this path with the vertex v_f, we get a path P between u and w in G^* which has no internal vertices in C'. Suppose P contains an internal vertex v with $v \in S'$. Observe that P must also contain $h(v)$ (if $h(v) = v$ then this is obvious, and otherwise v has only two neighbours, both of which must be in P and one of which is $h(v)$). If $h(v) \neq u$ and $h(v) \neq w$, then $h(v)$ is also an internal vertex of P, and P has an internal vertex in C', a contradiction. Therefore the only internal vertices v of P which are in S' are those for which $h(v) = u$ or $h(v) = w$.

If P does not have any vertices in $h^{-1}(u)$ (which may happen if $u \notin S'$), then u must have a neighbour v_f with $h(v_f) = u$. Then by adding such a neighbour to P, we may assume that P contains at least one vertex in $h^{-1}(u)$. Similarly we may assume P contains at least one vertex in $h^{-1}(w)$. By considering the shortest subpath of P containing vertices in both $h^{-1}(u)$ and $h^{-1}(w)$, we have that there is a path in G^* with endpoints $s,t \in S'$, with no internal vertices in S', such that $h(s) = u, h(t) = w$. It follows that s,t are adjacent in torso(G^*,S').

Now consider $\beta^{-1}(u)$ for some vertex $u \in C'$. We wish to show that $\beta^{-1}(u)$ forms a connected subgraph in $\mathscr{T}$. As $\beta^{-1}(u) = \bigcup\{\beta'^{-1}(v) : v \in h^{-1}(u)\}$, each $\beta'^{-1}(v)$ forms a connected subgraph in $\mathscr{T}$, and $\beta'^{-1}(v_1) \cap \beta'^{-1}(v_2) \neq \emptyset$ for adjacent v_1, v_2 in torso(G^*, S'), it will be sufficient to show that $h^{-1}(u)$ induces a connected subgraph in torso(G^*, S'). If $u \in S'$, then all vertices in $h^{-1}(u) \setminus \{u\}$ are adjacent to u in torso(G^*, S^*), and therefore $h^{-1}(u)$ induces a graph that contains a star rooted at u as a subgraph. On the other hand if $u \notin S'$, then for any $v_1, v_2 \in h^{-1}(u)$, there is a path $v_1 u v_2$ in G^*, which contains no internal vertices in S', and so v_1, v_2 are adjacent in torso(G^*, S'). Therefore $h^{-1}(u)$ induces a clique in torso(G^*, S'). In either case, $h^{-1}(u)$ induces a connected subgraph in torso(G^*, S'). This satisfies the third condition of the tree decomposition, which completes the proof that $(\mathscr{T}, \beta)$ is a tree decomposition of torso(G, C').

Finally, note that by construction $\max_{x \in V(\mathscr{T})}(|\beta(x)| - 1) \leq \max_{x \in V(\mathscr{T})}(|\beta'(x)| - 1)$, and so $(\mathscr{T}, \beta)$ has width at most $g(l', e')$, which as previously discussed is bounded by a function depending only on l and e.

It remains to analyse the running time. Construction of G^* can be done in linear time as we need to process each edge of G once. G^* has $2|E(G)|$ edges and $|V(G)| + |E(G)|$ vertices, and therefore the algorithm of Lemma 2.2.8 takes time $f(l', e') \cdot (|E(G^*)| + |V(G^*)|) \leq f(l, l+e) \cdot (3|E(G)| + |V(G)|) \leq 3f(l, l+e) \cdot (|E(G)| + |V(G)|)$. Finally, transforming the decomposition $(\mathscr{T}, \beta')$ into $(\mathscr{T}, \beta)$ takes time $O(|V(\mathscr{T})| \cdot \max_{x \in V(\mathscr{T})} |\beta(x)| = O(|V(\mathscr{T})| \cdot g(l, l+e)$, and we may assume $|V(\mathscr{T})|$ is linear in $|E(G)| + |V(G)|$ as it took linear time to construct. Therefore the total running time is linear in $|E(G)| + |V(G)|$. ∎

We will now use the treewidth result on torso graphs to construct a tree decomposition of the original graph, in which the width may not be bounded, but the intersection between bags and the number of edges in small cuts in each bag is bounded by a function of p. In order to make our dynamic programming simpler, it is useful to place further restrictions on the structure of a tree decomposition. The notion of a *nice tree decomposition* is often used in dynamic programming, as it can impose a simple structure and can be found whenever we have a tree decomposition.

Definition 2.2.6 *Given an undirected graph $G = (V, E)$, a* nice tree decomposition $(\mathscr{T}, \beta)$ *is a tree decomposition such that $\mathscr{T}$ is a rooted tree, and each of the nodes $x \in V(\mathscr{T})$ falls under one of the following classes:*

- *x* **is a Leaf node:** *Then x has no children in $\mathscr{T}$;*
- *x* **is an Introduce node:** *Then x has a single child y in $\mathscr{T}$, and there exists a vertex $v \notin \beta(y)$ such that $\beta(x) = \beta(y) \cup \{v\}$;*

- x **is a Forget node:** *Then x has a single child y in $\mathscr{T}$, and there exists a vertex $v \in \beta(y)$ such that $\beta(x) = \beta(y) \setminus \{v\}$;*
- x **is a Join node:** *Then x has two children y and z, and $\beta(x) = \beta(y) = \beta(z)$.*

(Note that sometimes it is also required that $|\beta(x)| = 1$ for every leaf node x, but for our purposes we allow $\beta(x)$ to be unbounded.)

It is well-known that given a tree decomposition of a graph, it can be transformed into a nice tree decomposition of the same width in polynomial time [54].

Lemma 2.2.10 (Lemma 13.1.3, [54]) *For constant k, given a tree decomposition of a graph G of width k and $O(n)$ nodes, where n is the number of vertices of G, one can find a nice tree decomposition of G of width k and with at most $4n$ nodes in $O(n)$ time.*

It is also known that a tree decomposition of a graph can be found in fixed-parameter time.

Lemma 2.2.11 *[16] There exists an algorithm that, given an n-vertex graph G and integer k, runs in time $k^{O(k^3)} \cdot n$ and either constructs a tree decomposition of G of width at most k, or concludes that G has treewidth greater than k.*

Observe that as the running time in Lemma 2.2.11 is $k^{O(k^3)} \cdot n$, we may assume the tree decomposition has at most $k^{O(k^3)} \cdot n$ nodes. Then applying Lemma 2.2.10, we have that for any graph G with treewdith k, we can find a nice tree decomposition of G with at most $4|V(G)|$ nodes in time fixed-parameter with respect to k.

Our tree decomposition will be similar but not identical to a nice tree decomposition. We are now ready to give our tree decomposition, which is the main result of this section.

We believe this lemma may be useful for other problems in which the "difficult" parts of a graph are the small cuts or separators.

Lemma 2.2.12 *Let a,b be vertices of a graph $G = (V,E)$ and let l be the minimum size of an (a,b)-cut (respectively, let l be the minimum size of an (a,b)-separator). For some $e \geq 0$, let D be the union of all minimal (a,b)-cuts of size at most $l+e$, and let $C = V(D) \setminus (a,b)$ (respectively, let C be the union of all minimal (a,b)-separators of size at most $l+e$).*

Then there is an $f(l,e) \cdot (|E| + |V|)$ time algorithm that returns a set C' disjoint from $\{a,b\}$ and a (binary) tree decomposition $(\mathscr{T}, \beta)$ of G such that:

1. *$C \subseteq C'$;*

2. *$\beta(x) \subseteq C'$ for any node x in $\mathscr{T}$ which is not a leaf node (in particular, the intersection between any two bags of adjacent nodes of $\mathscr{T}$ is contained in C');*
3. *For any node x in $\mathscr{T}$, $|\beta(x) \cap C'| \leq g(l,e)$;*
4. *$(\mathscr{T},\beta)$ restricted to C' (i.e. $(\mathscr{T},\beta')$, where $\beta'(x) = \beta(x) \cap C'$) is a nice tree decomposition;*

for some functions f and g depending only on l and e.

Proof If C is the union of all vertices appearing in the set D of all minimal (a,b)-cuts of size at most $l+e$, then apply Lemma 2.2.9. If C is the union of all minimal (a,b)-separators of size at most $l+e$, then apply Lemma 2.2.8. In either case, we get a set $C' \supseteq C$ disjoint from $\{a,b\}$ such that $\text{tw}(\text{torso}(G,C')) \leq g(l,e)$, for a function g depending only on l and e. From here on the proof is identical for the two cases.

Using Lemmas 2.2.10 and 2.2.11, we may find a nice tree decomposition of $\text{torso}(G,C')$ of width at most $g(l,e)$ in time $f(g(l,e)) \cdot (|E|+|V|)$, for some function f depending only on $g(l,e)$. Let $(\mathscr{T}',\beta')$ be the resulting tree decomposition of $\text{torso}(G,C')$.

We now add the vertices of G which are not in C' to this decomposition. Consider any component X of $G-C'$. Then $N(X) \subseteq C'$. Furthermore, by definition of $\text{torso}(G,C')$, any pair of vertices in $N(X)$ are adjacent in $\text{torso}(G,C')$. It is well-known that the vertices of a clique in a graph are fully contained in a single bag in any tree decomposition of the graph. Therefore, $N(X) \subseteq \beta'(x)$ for some node x in $\mathscr{T}$.

If x is a Leaf node then modify $\beta'(x)$ by adding X to it. Otherwise, modify $(\mathscr{T}',\beta')$ by inserting (in the edge of $\mathscr{T}$ between x and its parent) a new Join node y as the parent of x, with another child node z of y, such that $\beta'(y) = \beta'(x)$, and $\beta'(z) = \beta'(x) \cup X$. Thus, X is still added to a Leaf node.

Let $(\mathscr{T},\beta)$ be the resulting tree decomposition. As every component of $G-C'$ was added to a bag in a tree decomposition of $\text{torso}(G,C')$, $\bigcup_{x \in V(\mathscr{T})} \beta(x) = V(G)$. Every edge between vertices in C' is in a bag due to the tree decomposition of $\text{torso}(G,C')$, and for every $v \notin C'$, $N(v)$ is contained in the same bag as v. Therefore for every edge uv in G, u and v appear in the same bag. For any vertex v, $\beta^{-1}(v)$ consists of a single node if $v \notin C'$, and otherwise $\beta^{-1}(v)$ is connected by the tree decomposition of $\text{torso}(G,C')$. Thus, $(\mathscr{T},\beta)$ is a tree decomposition of G.

Furthermore, by construction $\beta(x) \subseteq C'$ for every non-leaf node x, $|\beta(x) \cap C'| \leq g(l,e)$ for every node x, and $(\mathscr{T},\beta)$ restricted to C' is a nice tree decomposition. ∎

We now modify this approach slightly to get the desired tree decomposition when C is the union of all edges in small t-cuts.

Lemma 2.2.13 *Let $(G=(V,E),w,t)$ be an instance of* BCPP*, let C be the non-empty set of vertices appearing in edges in small t-cuts. Then there is an $f(p)\cdot(|E|+|V|)$ time algorithm that returns a set C' and a (binary) tree decomposition $(\mathscr{T},\beta)$ of G such that:*

1. *$C\subseteq C'$;*
2. *$\beta(x)\subseteq C'$ for any node x in $\mathscr{T}$ which is not a leaf node (in particular, the intersection between any two bags of adjacent nodes of $\mathscr{T}$ is contained in C');*
3. *For any node x in $\mathscr{T}$, $|\beta(x)\cap C'|\leq g(p)$;*
4. *$(\mathscr{T},\beta)$ restricted to C' (i.e. $(\mathscr{T},\beta')$, where $\beta'(x)=\beta(x)\cap C'$) is a nice tree decomposition;*

for some functions f and g depending only on p.

Proof First construct the multigraph G^* from G by creating two new vertices a,b, with $t(v)$ edges between a and v for each $v\in V_t^+$, and $-t(v)$ edges between b and v for each $v\in V_t^-$. Then by definition, C is the set of vertices appearing in an edge $e\in E(G)$ such that e is part of a minimal (a,b)-cut in G^* of size less than p.

Now apply Lemma 2.2.12 to get a set C' disjoint from $\{a,b\}$ such that $C\subseteq C'$ and a tree decomposition of torso(G^*,C') with treewidth at most $g(l,e)$, where $l+e=p-1$ and so $g(l,e)$ is bounded by a function of p. It follows from the definition of a torso graph that torso$(G^*\setminus\{a,b\},C')$ is a subgraph of torso$(G^*,C')\setminus\{a,b\}$ [66, Lemma 2.6], and so we can get a tree decomposition $(\mathscr{T},\beta)$ of torso(G,C') by removing a and b from every bag in the tree decomposition of torso(G^*,C'). As $a,b\notin C'$, the resulting tree decomposition is still a nice tree decomposition when restricted to C'. ∎

2.2.5 Dynamic Programming

Let (G,w,t) be an instance of BCPP. Let $(\mathscr{T},\beta)$ be the tree decomposition of G and C' the set of vertices containing all vertices of every small t-cut given by Lemma 2.2.13. In this section we give a dynamic programming algorithm based on this decomposition.

Before describing the algorithm, we give some notation that we use in this section. Let $\alpha(x)=\beta(x)\cap C'$. Thus $\beta(x)\cap\beta(y)\subseteq\alpha(x)$ for all nodes $x\neq y$, and $\alpha(x)=\beta(x)$ for every non-leaf x. Furthermore, for any Join node x with two children y and z, we have that $\alpha(x)=\alpha(y)=\alpha(z)$, even if one or both of the children of x is a Leaf node whose bag contains vertices not in C'. Let $\gamma(x)$ be the union of the bags of all descendants of x including x itself. Thus, if r is the root node of $\mathscr{T}$, then $\gamma(r)=V(G)$.

Let $h : V(G) \to \{\text{ODD}, \text{EVEN}\}$ be the function such that $h(v) = \text{ODD}$ if $t(v)$ is odd and $h(v) = \text{EVEN}$ if $t(v)$ is even. Observe that in the undirected version of any solution to BCPP on (G,w,t), each vertex v will have odd degree if and only if $h(v) = \text{ODD}$. Thus, h and similar functions will be used to tell us whether a vertex should have odd or even degree.

To simplify some expressions, we adopt the convention that $\text{ODD} + \text{ODD} = \text{EVEN}, \text{EVEN} + \text{EVEN} = \text{EVEN}$, and $\text{ODD} + \text{EVEN} = \text{ODD}$. We say a vertex v is *h-balanced* if it has odd degree if and only if $h(v) = \text{ODD}$. An undirected multigraph H is *h-balanced* if every vertex is h-balanced.

We now give an outline of our algorithm.

By Corollary 2.2.1, in order to solve an instance (G,w,t) of the BCPP, it is enough to find an undirected multigraph H of minimum weight, such that the underlying graph of H is G, H has a t-road, and $d_H(v) - t(v)$ is even for every vertex v. Our algorithm will therefore focus on solving this problem, rather than finding an optimal multi-orientation of G directly. By Lemma 2.2.7, we may assume H has a well-(G,t)-behaved t-road T. Our dynamic programming algorithm will give a way to find H and T.

For each node x in $\mathscr{T}$, our dynamic programming algorithm will calculate a value $\psi(x,H',T',t',h')$, for a particular range of graphs H' and T' and functions t' and h'. Informally, $\psi(x,H',T',t',h')$ denotes a potential solution of minimum weight, restricted to $\gamma(x)$. Let H denote this restricted solution and T its "t-road", similarly restricted to $\gamma(x)$. The subgraph H' tells us how H should look when restricted to $\alpha(x)$, and similarly T' tells us how T should look when restricted to $\alpha(x)$. The function t' tells us what the imbalance of each vertex should be within T. Roughly speaking, it is the function for which T is a t'-road. (Note that T will not necessarily be a t-road itself, as it is only a restriction of a potential t-road to $\gamma(x)$.) In a similar way, the function h', which maps vertices to either ODD or EVEN, tells us whether the degree of each vertex within H should be odd or even. (We note that in a full solution, the parity of the degree of each vertex v will be defined by $h(v)$, but as H is only a partial solution it may be that $h'(v) \neq h(v)$ for some v.)

More formally, let x be a node of $\mathscr{T}$, let H' be an undirected multigraph with underlying graph $G[\alpha(x)]$, such that $\mu_{H'}(uv) \leq \max\{p,2\}$ for all edges uv. Let T' be a directed graph with vertex set $\alpha(x)$, such that $\mu_{T'}(\overrightarrow{uv}) + \mu_{T'}(\overrightarrow{vu}) \leq \mu_{H'}(uv)$ for all edges uv. Let t' be a function $\alpha(x) \to [-p,p]$ and let h' be a function $\alpha(x) \to \{\text{ODD}, \text{EVEN}\}$. Then let $\psi(x,H',T',t',h')$ be an undirected multigraph H with underlying graph $G[\gamma(x)]$, of minimum weight such that

1. $H[\alpha(x)] = H'$.

2. H has a well-(G,t)-behaved t^*-road T such that T restricted to $\alpha(x)$ is T', where $t^* : \gamma(x) \to [-p, p]$ is the function such that $t^*(v) = t'(v)$ for $v \in \alpha(x)$ and $t^*(v) = t(v)$, otherwise.

3. H is h^*-balanced, where $h^* : \gamma(x) \to \{\text{ODD}, \text{EVEN}\}$ is the function such that $h^*(v) = h'(v)$ if $v \in \alpha(x)$ and $h^*(v) = h(v)$, otherwise.

If no such H exists, let $\psi(x, H', T', t', h') = null$.

The following lemma shows that to solve the BCPP, it is enough to calculate $\psi(x, H', T', t', h')$ for every choice of x, H', T', t', h'.

Lemma 2.2.14 *Let r be the root node of $\mathscr{T}$. Let t' be t restricted to $\alpha(r)$, and let h' be h restricted to $\alpha(r)$. Let H' and T' be chosen such that the weight of $H = \psi(r, H', T', t', h')$ is minimized. Then the weight of H is the weight of an optimal solution to the* BCPP *on (G, w, t), and given H we may construct an optimal solution to* BCPP *on (G, w, t) in polynomial time.*

Proof Observe that by construction of t' and h', t^* and h^* in the definition of $\psi(r, H', T', t', h')$ are t and h, respectively. Also observe that for a graph H, $d^*_H(v) - t(v)$ is even for each vertex v if and only if H is h-balanced.

Let D be an optimal solution to the BCPP on (G, w, t), and let H be the undirected version of D. By Lemma 2.2.4, H is h-balanced and has a t-road. Furthermore by Lemma 2.2.7, H has a well-(G,t)-behaved t-road T. By Lemma 2.2.3, we may assume that $\mu_H(uv) \leq \max\{p, 2\}$ for each edge uv. H clearly has underlying graph $G = \gamma(r)$. So by letting H' be $H[\alpha(r)]$ and letting T' be $T[\alpha(r)]$, we have that H satisfies all the conditions of $\psi(x, H', T', t', h')$ (except possibly for minimality).

On the other hand, suppose H satisfies all these conditions. Then in particular, H has underlying graph $\gamma(r) = G$, H is h-balanced, and H has a t-road. It follows by Lemma 2.2.4 that there exists a solution to the BCPP on (G, w, t) which is an orientation of H.

It follows that the minimum weight solution to the BCPP on (G, w, t) has the same weight as $H = \psi(r, H', T', t', h')$, when H' and T' are chosen such that the weight of H is minimized. ∎

When using dynamic programming algorithms based on tree decompositions, the most commonly used approach is to consider the restriction of possible solutions to each bag, and combine information about the possible restrictions on each bag to construct a full solution. However this approach only works when the tree decomposition is of bounded width, as the number of restrictions to consider on each bag is bounded. In our case, some of the bags in the decomposition may be arbitrarily large, so we cannot consider all possible solutions

on a bag. However, we do have that each bag contains a bounded number of vertices from C', where C' contains all vertices that appear in edges in small t-cuts. It will turn out to be enough to make a guess based on the edges between vertices in C', after which the rest of the problem can be solved efficiently.

Finally, we show how to calculate $\psi(x,H',T',t',h')$ for every choice of x,H',T',t',h'.

Lemma 2.2.15 *$\psi(x,H',T',t',h')$ can be calculated in FPT time, for all choices of x,H',T',t',h'.*

Proof Consider some node x, and assume that we have already calculated $\psi(y,H'',T'',t'',h'')$, for all descendants y of x and all choices of H'',T'',t'',h''. We consider the possible types of nodes separately.

x **is a Leaf node:** Consider the multigraph G_x with vertex set $\beta(x) = \gamma(x)$ such that $G_x[\alpha(x)] = H'$, and G_x has exactly one copy of each edge in $G[\beta(x)]$ not contained in $\alpha(x)$. Note that G_x is necessarily a subgraph of $H = \psi(x,H',T',t',h')$. Moreover, as every edge from $G[\beta(x)]$ in a small t-cut of G is contained in $\alpha(x)$, any well-(G,t)-behaved t^*-road in H is also a well-(G,t)-behaved t^*-road in G_x.

It follows that if there exists any H satisfying the first two conditions of $\psi(x,H',T',t',h')$ then G_x has a well-(G,t)-behaved t^*-road. So we first check whether G_x has a t^*-road (we note that any t^*-road in G_x is well-(G,t)-behaved by construction of G_x).

If G_x has a t^*-road, it remains to find a minimum weight (multi)set of edges to add to G_x to make it h^*-balanced. This can be done by solving the MINIMUM WEIGHT X-JOIN PROBLEM, where X is the set of all vertices in $\beta(x)$ that are not h^*-balanced in G_x. By Lemma 2.2.5, this can be done in polynomial time.

x **is an Introduce node:**

Let y be the child node of x, and let v be the single vertex in $\beta(x) \setminus \alpha(y)$. Then no vertices in $\gamma(x)$ are adjacent with v, except for those in $\alpha(x)$. In particular for any $H = \psi(x,H',T',t',h')$, the only edges of H incident with v are those in H'. Thus, if v is not h'-balanced in H', then $\psi(x,H',T',t',h') = \text{NULL}$. Similarly, if v is not t'-balanced in T', then $\psi(x,H',T',t',h') = \text{NULL}$.

Otherwise, suppose that $H = \psi(x,H',T',t',h')$, and let H^* be H restricted to $\gamma(y)$. We now construct the values H'',T'',t'',h'' for which $H^* = \psi(y,H'',T'',t'',h'')$ must hold. Observe that $H^*[\alpha(y)] = H[\alpha(y)] = H'[\alpha(y)]$. Thus we will set $H'' = H'[\alpha(y)]$. For the well-(G,t)-behaved t'-road T that H must have, let T^* be T restricted to $\gamma(y)$, and observe that $T^*[\alpha(y)] = T[\alpha(y)] = T'[\alpha(y)]$. Thus we will set $T'' = T'[\alpha(y)]$. As T^* is equal to T with the arcs incident to u removed, we have that the imbalance of a vertex $u \in \alpha(y)$ in T^* is $t'(u) - \mu_{T'}(\overrightarrow{uv}) + \mu_{T'}(\overrightarrow{vu})$. (Note also that for $u \in \gamma(y) \setminus \alpha(u)$, the imbalance of u remains unchanged i.e. is still $t(u)$.) Thus we let $t'' : \alpha(y) \to [-p,p]$ be the function such

that $t''(u) = t'(u) - \mu_{T'}(\overrightarrow{uv}) + \mu_{T'}(\overrightarrow{vu})$. By a similar argument, the parity of the degree of any vertex $u \in \alpha(y)$ in H^* is equal to $h''(u)$, where $h'' : \alpha(y) \to \{\text{ODD}, \text{EVEN}\}$ is such that if $\mu_{H'}(uv)$ is odd then $h''(u) = h'(u) + \text{ODD}$, and otherwise $h''(u) = h'(u)$. Thus, we have that $H^* = \psi(y, H'', T'', t'', h'')$ (where the minimality property follows from the fact that any improvement on H^* would give a corresponding improvement on H),

Thus, we may set $\psi(x, H', T', t', h')$ to be $\psi(y, H'', T'', t'', h'')$ together with the edges of H' incident with v.

x **is a Forget node:**

Let y be the child node of x, and let v be the single vertex in $\alpha(y) \setminus \beta(x)$. Note that $\gamma(y) = \gamma(x)$. Let $t'' : \alpha(y) \to [-p, p]$ be the function t^* restricted to $\alpha(y)$ (i.e. t'' extends t' and assigns v to $t(v)$). Let $h'' : \alpha(y) \to \{\text{ODD}, \text{EVEN}\}$ be the function h^* restricted to $\alpha(y)$ (i.e. extends h' and assigns v to $h(v)$). If $H = \psi(x, H', T', t', h')$ then by construction of t'' and h'', we also have $H = \psi(y, H'', T'', t'', h'')$, for some values of H'' and T''. Note that the possible values of H'' are those for which $H''[\alpha(x)] = H'$, and so the only choice is the multiplicity of each edge in H'' incident with v. By Lemma 2.2.3 we may assume the multiplicity of each such edge is at most p, and therefore we have at most $(p+1)^{|\alpha(x)|} \leq (p+1)^{g(p)}$ possible values of H''. Similarly, we have at most $(p+1)^{|\alpha(x)|} \leq (p+1)^{2g(p)}$ possible values of T''.

We therefore may set $\psi(x, H', T', t', h')$ to be the minimum weight $\psi(y, H'', T'', t'', h'')$ over all possible values of H'' and T''.

x **is a Join node:**

Let y and z be the children of x, and recall that $\alpha(x) = \alpha(y) = \alpha(z)$, and furthermore $\gamma(x) = \gamma(y) \cup \gamma(z)$ and $\gamma(y) \cap \gamma(z) = \alpha(x)$.

Suppose that $H = \psi(x, H', T', t', h')$, and consider the graphs $H_y = H[\gamma(y)]$ and $H_z = H[\gamma(z)]$.

Observe that the weight of H is equal to $w(H_y) + w(H_z) - w(H')$ (as the only edges contained in both H_y and H_z are those in H'). Since $\alpha(x) = \alpha(y)$, it must be the case that $H_y = \psi(y, H', T', t'', h'')$ for some choice of t'' and h''. Similarly $H_z = \psi(z, H', T', t''', h''')$ for some choice of t''' and h'''. It remains to determine the possible choices of t'', h'', t''', h'''.

Consider a well-(G,t)-behaved directed multigraph T, and let $T_y = T[\gamma(y)]$ and $T_z = T[\gamma(z)]$. Let $t^{*\prime\prime}$ be the function such that T_y is a $t^{*\prime\prime}$-road, and let $t^{*\prime\prime\prime}$ be the function such that T_z is a $t^{*\prime\prime\prime}$-road. For any $v \in \alpha(x)$, the imbalance of v in T is equal to $t^{*\prime\prime}(v) + t^{*\prime\prime\prime}(v) - \sum_{u \in \alpha(x)} \mu_{T'}(\overrightarrow{vu}) + \sum_{u \in \alpha(x)} \mu_{T'}(\overrightarrow{uv})$ (where the last two terms come from the fact that arcs in T' are counted twice in $t^{*\prime\prime}(v) + t^{*\prime\prime\prime}(v)$). Thus, v is t^*-balanced in T if and only if $t^*(v) = t^{*\prime\prime}(v) + t^{*\prime\prime\prime}(v) - \sum_{u \in \alpha(x)} \mu_{T'}(\overrightarrow{vu}) + \sum_{u \in \alpha(x)} \mu_{T'}(\overrightarrow{uv})$. We also note that for $v \in \gamma(y) \setminus \alpha(x)$, v is t^*-balanced in T if and only if $t^{*\prime\prime}(v) = t^*(v) = t(v)$, and for $v \in \gamma(z) \setminus \alpha(x)$, v is t^*-balanced in T if and only if $t^{*\prime\prime\prime}(v) = t^*(v) = t(v)$.

Let $h^{*\prime\prime} : \gamma(y) \to \{\text{ODD}, \text{EVEN}\}$ be the function such that H_y is $h^{*\prime\prime}$-balanced, and let $h^{*\prime\prime\prime} : \gamma(z) \to \{\text{ODD}, \text{EVEN}\}$ be the function such that H_z is $h^{*\prime\prime\prime}$-balanced. Then by a similar argument, a vertex $v \in \alpha(x)$ is h^*-balanced in H if and only if $h^*(v) = h^{*\prime\prime}(v) + h^{*\prime\prime\prime}(v) + c$, where $c = \text{EVEN}$ if v has even degree in H, and $c = \text{ODD}$ otherwise.

The above implies that $\psi(x, H', T', t', h')$ is the union of $\psi(y, H', T', t'', h'')$ and $\psi(z, H', T', t''', h'')$, where t'', h'', t''', h''' are chosen to minimize the the total weight of $\psi(y, H', T', t'', h'')$ and $\psi(z, H', T', t''', h'')$ and such that

1. $t''(v), t'''(v) \in [-p, p]$ for all $v \in \alpha(x)$;
2. $t'(v) = t''(v) + t'''(v) - \sum_{u \in \alpha(x)} \mu_{T'}(\overrightarrow{vu}) + \sum_{u \in \alpha(x)} \mu_{T'}(\overrightarrow{uv})$ for all $v \in \alpha(v)$;
3. $h'(v) = h''(v) + h'''(v)$ if v has even degree in H, and $h'(v) = \text{ODD} + h''(v) + h'''(v)$ otherwise.

Observe that in the case of a Join node, there is only one possible choice of t''' for each choice of t'' and only one possible choice of h''' for each choice of h''. Therefore there are at most $[2(2p+1)]^{g(p)}$ possible choices for t'', t''', h'', h'''. Therefore it is possible to calculate $\psi(x, H', T', t', h')$ in fixed-parameter time, as long as we have already calculated $\psi(y, H'', T'', t'', h'')$, for all descendants y of x and all choices of H'', T'', t'', h''.

It remains to show that the number of graphs $\psi(x, H', T', t', h')$ to calculate is bounded by a function of p times a polynomial in $|V(G)|$. We may assume the number of nodes x in $\mathscr{T}$ is bounded by $|V(G)|$. As $|\alpha(x)|$ is bounded by a function of p, and H' has at most $\max\{p, 2\}$ edges for each edge within $\alpha(x)$, and T' has at most $\max\{p, 2\}$ arcs for each edge within $\alpha(x)$, the number of possible graphs H' and T' is bounded by a function of p. Finally, as $|\alpha(x)|$ is bounded by a function of p, the number of possible functions $t : \alpha(x) \to [-p, p]$ and $h' : \alpha(x) \to \{\text{ODD}, \text{EVEN}\}$ is also bounded by a function of p. ∎

Lemmas 2.2.14 and 2.2.15 imply the following:

Theorem 2.2.1 BCPP *is fixed-parameter tractable.*

Theorem 2.2.1 and Lemma 2.2.2 imply the following:

Theorem 2.2.2 k-ARC CPP *is fixed-parameter tractable.*

2.3 Chinese Postman Problem on Edge Colored Graphs

Recall that a PC Euler trail in a multigraph G is a properly colored closed walk which traverses each edge of G exactly once. PC Euler trails were one of the first types of PC walks studied in the literature and the first papers that studied PC Euler trails were motivated by theoretical questions [34, 57] as well as questions in molecular biology [76]. To formulate a characterization of edge-colored graphs with PC Euler trails by Kotzig [57], we introduce some additional terminology. A vertex in an edge-colored multigraph is *balanced* if no color appears on more than half of the edges incident with the vertex, and *even* if it is of even degree. We say that an edge-colored graph is *PC Eulerian* if it contains a PC Euler trail.

Theorem 2.3.1 *An edge-colored multigraph G is PC Eulerian if and only if G is connected and every vertex of G is balanced and even.*

Benkouar *et al.* [15] described a polynomial-time algorithm to find a PC Euler trail in an edge-colored multigraph, if it contains one. Studying DNA physical mapping, Pevzner [77] came up with a simpler polynomial-time algorithm solving the same problem.

In this section, we consider the Chinese Postman Problem on edge-colored graphs (CPP-ECG) defined as following. Note that the weight of a walk is the sum of the weights of its edges.

CHINESE POSTMAN PROBLEM ON EDGE-COLORED GRAPHS (CPP-ECG)
Instance: A connected edge colored graph $G = (V,E)$, with vertex set V, edge set E; a weight function $w : E \to \mathbb{N}_0$.
Output: A PC closed walk in G which traverses all edges of G at least once and has the minimum weight among such walks.

Observe that to solve CPP-ECG, it suffices to find a PC Eulerian edge-colored multigraph G^* of minimum weight such that $V(G^*) = V(G)$ and for every pair of distinct vertices u, v and color i, G^* has $p^* > 0$ parallel edges between vertices u and v of color i if and only if G has at least one and at most p^* edges of color i between u and v. (To find the actual walk, we can use the algorithm from [15] or [76].)

CPP-ECG is a generalization of the PC Euler trail problem as an instance G has a PC Euler trail if and only if $G^* = G$. CPP-ECG is also a generalization of the Chinese Postman Problem (CPP) on both undirected and directed multigraphs (the arguments are the same as for PC walks above). However, while CPP on both undirected and directed multigraphs has a solution on every connected multigraph G, it is not the case for CPP-ECG. Indeed, there is no solution on any connected edge-colored multigraph containing a vertex incident to edges of only one color.

As we mentioned earlier in subsection 1.5, there are polynomial time algorithms for the Chinese Postman problem on both undirected and directed multigraphs. We will prove that CPP-ECG is polynomial-time solvable as well. Note that our proof is significantly more complicated than that for CPP on undirected and directed graphs. As in the undirected case, we construct an auxiliary edge-weighted complete graph H and seek a minimum-weight perfect matching M in it. However, the construction of H and the arguments justifying the appropriate use of M are significantly more complicated. This can partially be explained by the fact that CPP-ECG has no solution on many edge-colored multigraphs.

Note that there is another generalization of CPP on both undirected and directed multigraphs, namely, CPP on mixed multigraphs, i.e., multigraphs that may have both edges and arcs. However, CPP on mixed multigraphs is NP-hard [72]. It is fixed-parameter tractable when parameterized with both the number of edges and arcs [102, 42] and W[1]-hard when parameterized with pathwidth [46].

2.3.1 Preliminaries

For technical reasons we will consider walks with fixed end vertices and call them *fixed end-vertex (FEV) walks*. Note that an open walk is necessarily an FEV walk since the end-vertices are predetermined, whereas any vertex in a closed walk can be viewed as its two end-vertices and thus fixing such a vertex is somewhat similar to assigning a root vertex in a tree. An FEV walk $W = v_1e_1v_2\dots v_{p-1}e_{p-1}v_p$ is *PC* in an edge-colored graph if the colors of e_i and e_{i+1} are different for every $i \in [p-2]$. Note that we do not require that colors of e_{p-1} and e_1 are different even if $v_1 = v_p$. Thus, a PC FEV walk might not be a PC walk if $v_1 = v_p$.

Let $e = xy$ be an edge in an edge-colored multigraph G. The operation of *double subdivision* of e replaces e with an (x,y)-path P_e with three edges such that the weight of P_e equals that of edge e.

It is easy to see that in our study of PC walks, we may restrict ourselves to graphs rather than multigraphs. Indeed, it suffices to double subdivide every parallel edge e and assign the original color of e to the first and third edges of P_e and a new color to the middle edge.

Finding PC FEV walks. Let $\mathbb{R}_+$ denote the set of non-negative real numbers. To give our polynomial-time algorithm for CPP-ECG, we will use the following lemma:

Lemma 2.3.1 *Let $G = (V,E)$ be a k-edge-colored graph and $\omega : E \to \mathbb{R}_+$ a weight function. Let vertices $u, v \in V$ and edge colors c_1, c_2 be given, where we may have $u = v$. In polynomial*

time we can find a minimum-weight PC FEV walk from u to v in G whose first edge has color c_1 and whose last edge has color c_2, or conclude that there is no such PC FEV walk in G.

Proof Define an auxiliary digraph H as follows. Let the vertex set of H be $\{(u,0)\} \cup \{(x,i) : x \in V, i \in [k]\}$. For every edge $xy \in E$, of color i, we add to H all arcs $(x,j)(y,i)$ and $(y,j)(x,i)$ where $j \in [k]$, $j \neq i$. We also add an arc from $(u,0)$ to (z,c_1) for every edge $uz \in E$ of color c_1. Every arc in H retains the weight of the corresponding edge in G. We claim that the minimum-weight PC FEV walk we seek in G corresponds to a minimum-weight directed path from $(u,0)$ to (v,c_2) in H, which can be found in polynomial time, e.g., using Dijkstra's algorithm.

On the one hand, let $(x_1,d_1)(x_2,d_2)\dots(x_\ell,d_\ell)$ be a directed path in H such that $(x_1,d_1) = (u,0)$ and $(x_\ell,d_\ell) = (v,c_2)$. Note that in our construction of H, if there is an arc from (x,i) to (y,j), then $i \neq j$, moreover, it implies there is an edge between x and y in G of color j. Then by construction, $x_1e_2x_2\dots e_\ell x_\ell$, where e_i is an edge between x_{i-1} and x_i of color d_i, is a PC FEV walk in G with required properties. On the other hand, consider a minimum-weight PC FEV walk W in G with the properties requested. Orient the edges of the walk away from u. We may assume that no vertex has two in-coming directed edges in the walk of the same color, as the walk could otherwise be shortened. As above it is not hard to verify that the walk corresponds to a directed path P in H from $(u,0)$ to (v,c_2). By construction, the weight of P equals that of W. It remains to observe that P is a minimum-weight directed path from $(u,0)$ to (v,c_2), as otherwise there is a PC FEV walk between u and v with required edge colors of weight smaller than W, a contradiction.

Finally, we observe that the construction works without modification if $u = v$. ∎

2.3.2 Main Result

We are now ready to prove our main result.

Theorem 2.3.2 *There is a polynomial time algorithm for CPP-ECG.*

Proof Let G be a connected k-edge-colored graph with at least one edge. We consider CPP-ECG on G. We may assume that there is no vertex of G that is incident with edges of a single color only (As there is no PC closed walks G passing through such vertices). We may also assume that $k \geq 2$ is odd (if not, we double subdivide an edge e of G and assign a new color to the middle edge of P_e and the original color of e to the other two edges).

For a vertex $u \in V(G)$ and color $i \in [k]$, let $d_i(u)$ be the number of edges incident with u of color i. Let $d(u)$ $(= \sum_{i=1}^{k} d_i(u))$ be the degree of u in G. We say that color i is *dominant*

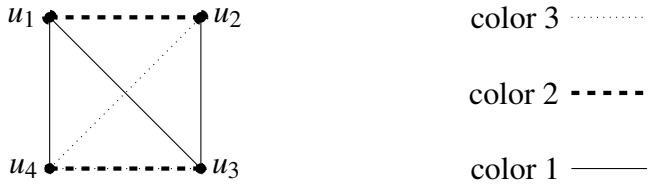

Fig. 2.2 3-edge-colored graph G

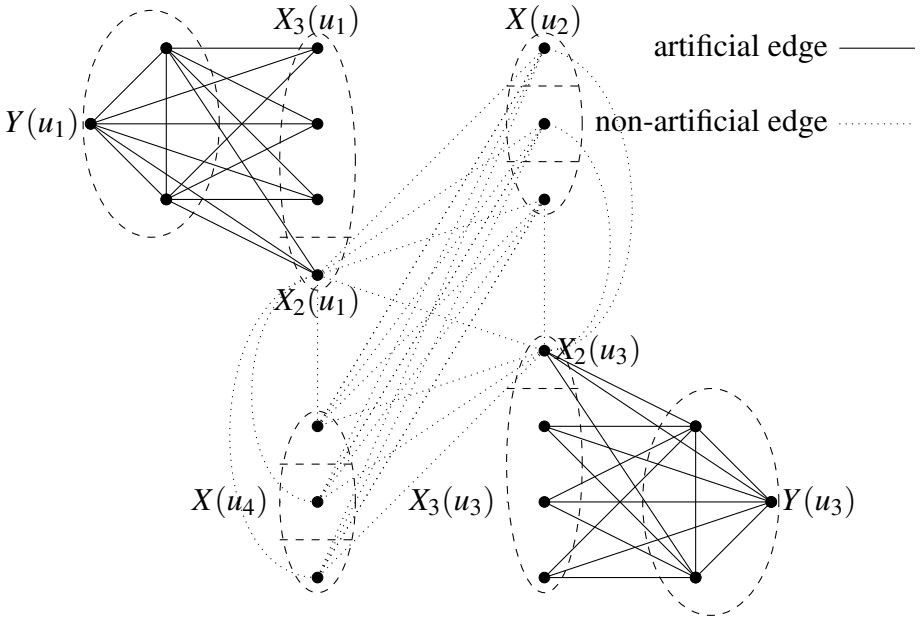

Fig. 2.3 Constructed graph H from the graph of Figure 2.2. Some edges, including artificial edges within $X(u_2)$ and $X(u_4)$, are omitted for clarity.

for u in G if $2d_i(u) > d(u)$; note that a vertex has at most one dominant color, and is balanced if and only if it has no dominant color.

We now show how to construct, in polynomial time, an undirected graph H such that H has a perfect matching if and only if G has a PC closed walk traversing all edges of G, and the minimum weight of a perfect matching in H is equal to the minimum weight of such a walk in G minus the weight of G. As computing a minimum-weight perfect matching can be done in polynomial time, the claim follows. An example is shown in Figures 2.2 and 2.3.

We will build the undirected graph H as follows. Define $\theta_i(u)$ as follows:

$$\theta_i(u) = \max\{0, d(u) - 2d_i(u)\}.$$

Let $X_i(u)$ be a set of independent vertices of size $\theta_i(u)$ and let $X(u) = \bigcup_{i=1}^{k} X_i(u)$. We now consider the cases when u is balanced and when u is not balanced, separately.

Case 1: u is not balanced. Let $Y(u)$ be a set of $(k-2)d(u)$ independent vertices and add all possible edges between $Y(u)$ and $X(u)$ and all possible edges within $Y(u)$. Let the weight of all these edges be zero and let us call them *artificial* edges. Let $Z(u) = X(u) \cup Y(u)$.

Case 2: u is balanced. Add all possible edges within $X(u)$. Let the weight of all these edges be zero and let call them *artificial* edges. Let $Z(u) = X(u)$.

For every pair $a \in X_i(u)$ and $b \in X_j(v)$ of distinct vertices such that ab is not an artificial edge, $i, j \in [k]$ and $u, v \in V(G)$ (we may have $i = j$ and/or $u = v$), add an edge between a and b with the weight equal to that of a minimum-weight PC FEV walk from u to v in G, starting in color i and ending in color j if one exists, otherwise add no edge ab; this can be computed in polynomial time by Lemma 2.3.1. This completes the description of H.

Assume that H has a perfect matching and M is a minimum-weight perfect matching in H. We will show that the weight of M plus the weight of all edges in G is the weight of an optimal solution to the CPP-ECG instance.

We begin with an observation about the structure of M.

Claim 2.3.1 *We say that a vertex $a \in X(u)$ for some $u \in V(G)$ is affected by M if a is incident with a non-artificial edge in M. Then M has the following properties.*

1. *If u is unbalanced with dominant color $i \in [k]$, then at least $2d_i(u) - d(u)$ vertices of $X(u)$ are affected by M. Furthermore $X_i(u) = \emptyset$.*
2. *If $d(u)$ is odd, then an odd number of vertices of $X(u)$ are affected by M.*
3. *If $d(u)$ is even, then an even number of vertices of $X(u)$ are affected by M.*

Furthermore, for any matching M_0 in H with no artificial edges that has the above properties, M_0 can be completed to a perfect matching by adding artificial edges.

Proof *1.* Assume that u is unbalanced with dominant color i, i.e., $2d_i(u) > d(u)$, $i \in [k]$. Then necessarily, $2d_j(u) < d(u)$ for every other color $j \in [k]$, hence $\theta_j(u) = d(u) - 2d_j(u)$ if $i \neq j$, whereas $\theta_i(u) = 0$. Thus

$$|X(u)| = \sum_{j \neq i} (d(u) - 2d_j(u)) = (k-2)d(u) + (2d_i(u) - d(u)),$$

where the last equality uses $\sum_{j \neq i} d(u) = (k-1)d(u)$ and $\sum_{j \neq i} d_j(u) = d(u) - d_i(u)$. Artificial edges on $X(u)$ can only match vertices of $X(u)$ against $Y(u)$. Since $|Y(u)| = (k-2)d(u) < |X(u)|$, this leaves at least $|X(u)| - |Y(u)| = 2d_i(u) - d(u)$ vertices in $X(u)$ which must be affected by M. Finally, $X_i(u) = \emptyset$ since $\theta_i(u) = 0$.

2 and 3. Assume first that u is unbalanced, so there is a set of vertices $Y(u)$. Since k is odd, the parity of $|Y(u)|$ matches the parity of $d(u)$, hence an odd (resp. even) number of vertices of $X(u)$ are matched against $Y(u)$ if and only if $d(u)$ is odd (resp. even). Now, as calculated in the previous paragraph, $|X(u)| = (k-3)d(u) + 2d_i(u)$, which is always even. Furthermore, the affected vertices of $X(u)$ are exactly those not matched against $Y(u)$. The claim follows.

Finally, if u is balanced, then $|X(u)| = \sum_j (d(u) - 2d_j(u)) = (k-2)d(u)$, which again has the same parity as $d(u)$. Since every artificial edge matches two vertices of $X(u)$, and the remaining vertices are exactly the affected vertices in $X(u)$, the claim follows.

Completing a non-perfect matching. Let M_0 be a matching in H such that for every vertex u, (1) if u has a dominant color i, then at least $2d_i(u) - d(u)$ vertices of $X(u)$ are affected by M_0, and (2) an odd number of vertices of $X(u)$ are affected by M_0 if and only if $d(u)$ is odd. By the above, the second point here implies that the number of unmatched vertices of $Z(u)$ is even for every vertex u. If u is balanced, then $Z(u) = X(u)$ is a clique and we can add artificial edges from the clique. If u is unbalanced, then $Y(u)$ is entirely unmatched, and by the first point here, the number of unmatched vertices in $X(u)$ is at most $|Y(u)|$. Hence the completion is possible. ∎

Let u and v be vertices of G and let i and j be colors. Let $e = ab$ be an arbitrary non-artificial edge in H, with $a \in X_i(u)$ and $b \in X_j(v)$, where we may have $u = v$ and/or $i = j$. An *e-walk* is a PC FEV walk in G, starting at u with an edge of color i, and ending at v with an edge of color j.

We will show the theorem in two parts. First, we will show that for any perfect matching M' of H, if we add an e-walk to G for every non-artificial edge $e \in M'$, then the resulting

multigraph G' is PC Euler. Here, to *add* an e-walk to G means to duplicate every edge along the walk, duplicating an edge multiple times if it occurs in the walk multiple times. Second, we show that for any PC Euler graph $G' = (V, E \cup W)$, obtained by duplicating edges in E, there exists a perfect matching M' of H such that W can be decomposed into a set $\mathscr{F}$ of e-walks, where there is an e-walk in $\mathscr{F}$ if and only if e is a non-artificial edge in M'. This will settle the result.

We need an observation about the effect of adding an e-walk to a graph.

Claim 2.3.2 *Let $e \in H$ be a non-artificial edge. Adding an e-walk F to G has the following effects.*

1. *For any vertex u, the parity of $d(u)$ changes if and only if F is open and u is either its first or last vertex.*
2. *If $u \in V(G)$ is neither the first nor the last vertex of the walk, then for every $i \in [k]$, the value of $d(u) - 2d_i(u)$ is non-decreasing in the process.*
3. *If F is closed, let u be its end-vertex, and let i (j, respectively) be the colors of its first (last, respectively) edges. Then for any $c \in [k]$, the value of $d(u) - 2d_c(u)$ increases by at least 2 if $c \notin \{i, j\}$; is non-increasing if $c \in \{i, j\}$ and $i \neq j$; and decreases by at most 2 if $i = j = c$.*
4. *If F is open, let u be an end-vertex of F, without loss of generality, the first one. Let i be the color of the first edge. Then for any $j \in [k]$, the value of $d(u) - 2d_j(u)$ decreases by at most one if $j = i$, and increases by at least one, otherwise.*

Proof The first item is easy. For the second item, we just observe that a single transition through u increases $d(u)$ by 2 and $d_i(u)$ by at most 1. Since the graph has no loops, the local effect on u of duplicating F decomposes into transitions, hence the second item holds. By the same argument, if u is an endpoint of F, then all visits to u except the first and/or last one decompose into transitions. This leaves only the first and last edges of F, and their effects on the end-vertices of F, to consider. The claims in items 3 and 4 follow by considering all possibilities for these two edges. ∎

We are now ready to show the first part of the theorem, as announced above. Let M' be an arbitrary perfect matching in H, and let G' be the result of adding an arbitrary e-walk, which has the same weight as edge e, to G for every non-artificial edge $e \in M'$. We will show that G' is PC Euler. By Theorem 2.3.1, we need to show three conditions: G' is connected, every vertex in G' is even, and every vertex in G' is balanced. The first condition follows since G

is connected; the second condition follows from Claim 2.3.1(2–3) and Claim 2.3.2(1). It remains to show that every vertex is balanced in G', i.e., for every $u \in V(G)$ and every $i \in [k]$, it holds in G' that $d(u) \geq 2d_i(u)$. We break this down into two cases.

Case 1: i is the dominant color for u in G. In this case, $d(u) - 2d_i(u) < 0$ in G, and we need to show that this value is nonnegative in G'. By Claim 2.3.1(1), at least $2d_i(u) - d(u)$ vertices of $X(u)$ are affected by M', and since $X_i(u) = \emptyset$, Claim 2.3.2 gives that the value of $d(u) - 2d_i(u)$ increases by at least 1 for every such vertex, and never decreases. Thus $d(u) \geq 2d_i(u)$ in G'.

Case 2: i is not a dominant color for u in G. In this case, $|X_i(u)| = d(u) - 2d_i(u) \geq 0$, and we need to show that this value is nonnegative in G'. By Claim 2.3.2, the value of $d(u) - 2d_i(u)$ decreases by at most as much as the number of vertices in $X_i(u)$ affected by M', in the sense of the term used in Claim 2.3.1. Since there are only $d(u) - 2d_i(u)$ such vertices, we see that $d(u) \geq 2d_i(u)$ also in G'.

Hence we conclude that $d(u) \geq 2d_i(u)$ in G' for every $u \in V(G)$ and every $i \in [k]$, hence every vertex is balanced. This concludes the proof that G' has a PC Euler trail. Clearly, the weight of this trail is equal to the total weight of $E(G)$ plus the sum of the weight of the added e-walks, where the latter part is exactly the weight of M'.

Now assume that CPP-ECG on G has a solution, a PC closed walk Q in G, and let G' be the graph obtained from G by replacing every edge $e = xy$ by q_e parallel edges with vertices x and y, where q_e is the number of times Q traverses e. Let $W = E(G') \setminus E(G)$, i.e., W are the edges that are added to G in order to get the PC Euler multigraph G'. We will find a perfect matching in H with total weight at most the sum of the weights of edges in W. This will complete the proof.

We initially define a set W' of walks as the set of one-edge walks xey, where $e = xy \in W$. We will merge walks in W' until we can map the remaining walks W' to a matching M_0 in H meeting the requirements of Claim 2.3.1, at which point we will be done. Here to merge two walks is to replace the walks $u_1e_1u_2 \dots e_{\ell-1}u_\ell$ and $v_1f_1v_2 \dots f_{h-1}v_h$, where $u_\ell = v_1 = u$, with the walk $u_1e_1u_2 \dots e_{\ell-1}uf_1v_2 \dots f_{h-1}v_h$. For $u \in V(G)$ and $i \in [k]$, let $w_i(u)$ denote the number of times that u is an end-vertex of a walk in W' and that walk ends in u with color i. Here we do not assume a fixed "first" and "last" vertex, and thus the walks $u_1e_1u_2 \dots e_{l-1}u_l$ and $u_le_{l-1}u_{l-1} \dots e_1u_1$ are the same. Note that we will allow walks in W' to start and end in the same vertex, in which case one walk may contribute to $w_i(u)$ twice.

By Claim 2.3.1, we need to ensure that

1. $w_i(u) \leq \theta_i(u) = |X_i(u)|$ for every $u \in V(G)$ and $i \in [k]$ (so that W' corresponds to a matching);

2. $\sum_{j\in[k]\setminus\{i\}} w_j(u) \geq 2d_i(u) - d(u)$ for every vertex u with a dominant color i in G; and

3. the parity condition is met for every vertex u.

Because G' has a PC closed walk traversing all edges, and by Theorem 2.3.1, we have that initially W' satisfies the following:

4. $\sum_{j\in[k]} w_j(u) + d(u) \geq 2w_i(u) + 2d_i(u)$ for every vertex $u \in V(G)$ and integer i (since G' is balanced);

5. $\sum_{j\in[k]} w_j(u)$ is even, if and only if $d(u)$ is even i.e. the parity condition is met (since G' is even, and hence $d(u) + \sum_{j\in[k]} w_j(u)$ is even).

We note that Condition 4 implies Condition 2 and Condition 5 implies Condition 3. As long as Condition 1 is not satisfied, we will modify W' by merging walks in such a way that Condition 4 and Condition 5 are still satisfied. As each merging reduces the number of walks in W', we must eventually stop with a set of walks W' satisfying Condition 1, Condition 2 and Condition 3.

So now assume that Condition 1 is not satisfied and let $u \in V(G)$ be a vertex such that $w_i(u) > \theta_i(u)$ for some $i \in [k]$. If $w_i(u) > d(u) - 2d_i(u)$, then we must have that $w_c(u) > 0$ for some $c \neq i$, as otherwise $2w_i(u) + 2d_i(u) > w_i(u) + d(u) = \sum_{j\in[k]} w_j(u) + d(u)$, a contradiction to Condition 4. Thus there are at least two colors c with $w_c(u) > 0$.

We will choose two colors h, j, with $w_h(u) > 0, w_j(u) > 0, h \neq j$ (i is not necessarily in $\{j,h\}$), and merge a walk ending at u with color h with a walk ending at u with color j. (If this makes us merge both end-vertices of the same walk, we may simply remove the walk). It is clear that the new walk is still PC, and this operation reduces the number of walks in W'. As we have reduced $w_h(u)$ and $w_j(u)$ by 1, and the other values are unaffected, it is clear that Condition 5 is still satisfied. We now show how to choose h, j in such a way that Condition 4 is still satisfied.

Let us call a color c *at risk* if $\sum_{j\in[k]} w_j(u) + d(u) \leq 2w_c(u) + 2d_c(u) + 1$ (informally, a color is "at risk" if removing two edges of other colors would lead that color to dominate u). As $\sum_{j\in[k]} w_j(u) + d(u)$ is necessarily even and $\sum_{j\in[k]} w_j(u) + d(u) \geq 2w_c(u) + 2d_c(u)$, we have that in fact $\sum_{j\in[k]} w_j(u) + d(u) = 2w_c(u) + 2d_c(u)$ for any at risk color c. Furthermore, we note that at most two colors in $[k]$ can be at risk. Indeed, suppose that distinct colors $c_1, c_2, c_3 \in [k]$ are at risk. Then $2w_{c_1}(u) + 2d_{c_1}(u) + 2w_{c_2}(u) + 2d_{c_2}(u) + 2w_{c_3}(u) + 2d_{c_3}(u) = 3(\sum_{j\in[k]} w_j(u) + d(u)) > 2(\sum_{j\in[k]} w_j(u) + d(u)) \geq 2(w_{c_1}(u) + w_{c_2}(u) + w_{c_3}(u) + d_{c_1}(u) + d_{c_2}(u) + d_{c_3}(u))$, a contradiction.

Next, suppose for a contradiction that $w_c(u) = 0$ for an at risk color c. Then $2d_c(u) = d(u) + \sum_{j\in[k]} w_j(u)$ and so $2d_i(u) \leq 2d(u) - 2d_c(u) = d(u) - \sum_{j\in[k]} w_j(u)$. Then $w_i(u) >$

$\theta_i(u) \geq d(u) - 2d_i(u) \geq d(u) - d(u) + \sum_{k \in [k]} w_j(u) \geq w_i(u)$, a contradiction. Thus, $w_c(u) > 0$ for any at risk color c.

We now know that there at least two colors c with $w_c(u) > 0$, there are at most 2 at risk colors, and if color c is at risk then $w_c(u) > 0$. We can therefore select two distinct colors h, j with $w_h(u) > 0, w_j(u) > 0$, such that any at risk color is contained in $\{h, j\}$. We now merge a walk ending with color h at u and a walk ending with color j at u, as described above. This has the effect of reducing each of $w_h(u)$ and $w_j(u)$ by 1, and leaving $w_c(u)$ unchanged for $c \in [k] \setminus \{h, j\}$. We now show that we still have $\sum_{j \in [k]} w_j(u) + d(u) \geq 2w_c(u) + 2d_c(u)$ for any $c \in [k]$. If $c \in \{h, j\}$, then both $\sum_{j \in [k]} w_j(u) + d(u)$ and $2w_c(u) + 2d_c(u)$ are reduced by 2, so the condition still holds. If $c \notin \{h, j\}$, then as c was not at risk we originally had $\sum_{j \in [k]} w_j(u) + d(u) \geq 2w_c(u) + 2d_c(u) + 2$. As $\sum_{j \in [k]} w_j(u) + d(u)$ is reduced by 2, we will still have $\sum_{j \in [k]} w_j(u) + d(u) \geq 2w_c(u) + 2d_c(u)$, as required.

We continue the above process until there is no $u \in V(G), i \in [k]$ for which Condition 1 fails. We therefore have that Conditions 1, 4 and 3 hold, which in turn implies Conditions 2 and 3 hold. Convert W' to a matching M_0 in H by adding for every walk F an edge e to M_0 such that F is an e-walk. This is possible since $w_i(u) \leq \theta_i(u) = |X_i(u)|$ for every $i \in [k]$, $u \in V(G)$. The weight of M_0 is at most the weight of W, since every edge e added to M_0 this way has a weight corresponding to a minimum-weight e-walk where F is just one possible e-walk. By Claim 2.3.1 we can complete M_0 to a perfect matching M' by adding artificial edges, which does not increase the weight. Hence H admits a perfect matching whose weight is at most the weight of W.

So in all cases we can find a perfect matching in H with weight exactly the weight of all the (non-closed) walks in W'. As we have already shown that a perfect matching in H gives rise to a solution to CPP-ECG on G where duplicated edges add the same weight as the weight of the matching, we are done. ∎

Chapter 3

Kernelization results on undirected graphs

3.1 Generalized Load Coloring Problem

Given a graph $G = (V,E)$ and an integer k, the 2-LOAD COLORING problem introduced in [3], asks whether there is a coloring $\varphi : V \to \{1,2\}$ such that for $i = 1$ and 2, there are at least k edges with both end-vertices colored i. The coloring needs not be proper. This problem was first proved to be NP-complete in [3], then Gutin and Jones studied its parameterized tractability with parameter k in [41]. They proved that 2-LOAD COLORING is fixed-parameter tractable by obtaining a kernel with at most $7k$ vertices. It is natural to extend 2-LOAD COLORING to c-LOAD COLORING as follows. Recall that, for a positive integer p, $[p] = \{1,2,\ldots,p\}$.

c-LOAD COLORING ((c,k)-LC)
Instance: A graph $G = (V,E)$ and two positive integers c and k.
Output: Decide whether there is a c-coloring $\varphi : V \to [c]$ such that for every $i \in [c]$, there are at least k edges with both end-vertices colored i.

If such a coloring φ exists, we call φ a (c,k)-coloring of G and we write $G \in (c,k)$-LC.

The c-LOAD COLORING problem can be viewed as a subgraph packing problem [61]: decide whether a graph G contains c vertex-disjoint k-edge subgraphs. Hence, $G \in (1,k)$-LC if and only if $|E(G)| \geq k$. In this chapter, we consider c-LOAD COLORING parameterized with k for every fixed $c \geq 2$. Note that c-LOAD COLORING is NP-complete for every fixed $c \geq 2$. Indeed, we can reduce 2-LOAD COLORING to c-LOAD COLORING with $c > 2$ by taking the disjoint union of G with $c-2$ stars $K_{1,k}$. Thus, G is a Yes-instance for 2-LOAD

COLORING if and only if the new graph is a Yes-instance for c-LOAD COLORING

We prove that the c-LOAD COLORING problem admits a kernel with less than $2ck$ vertices. Thus, for $c = 2$ we improve the kernel result of [41]. To show our result, we introduce reduction rules, which are new even for $c = 2$. We prove that the reduction rules can run in polynomial time and that an irreducible graph with at least $2ck$ vertices is in (c,k)-LC.

While there are many parameterized graph problems which admit kernels linear in the number of vertices, usually it is problems on classes of sparse graphs admit kernels linear in the number of edges (since in such graphs the number of edges is linear in the number of vertices), see, e.g., [17, 29, 62]. Only a few problems for general graphs are found to admit $O(k)$-edge kernels, see [49, 51, 79]. Our result is in the same category: c-LOAD COLORING admits a kernel with $O(k)$ edges for every fixed $c \geq 2$. Namely, the kernel has less than $6.25c^2k$ edges when $c \geq 2$ and, moreover, less than $6k + (3+\sqrt{2})\sqrt{k}$+4 edges when $c = 2$.

The optimization version of c-LOAD COLORING, called the MAX c-LOAD COLORING problem, is defined as follows.

MAX c-LOAD COLORING
Instance: A graph $G = (V,E)$ and a positive integer c.
Output: Decide the maximum integer k such that $G \in (c,k)$-LC.

Using the above bound on the number of edges in a kernel for $c \geq 2$, we show that MAX c-LOAD COLORING admits constant ratio approximation algorithms for any fixed c.

This section is organized as follows. After providing some additional terminology and notation in the remainder of this section, we show that the problem admits a kernel with less than $2ck$ vertices in section 3.1.1. Then, in section 3.1.2, we prove the upper bound on the number of edges in a kernel for every $c \geq 2$ and, in section 3.1.3, we show the constant ratio approximation result for MAX c-LOAD COLORING. We improve our bound for $c = 2$ in Section 3.1.4. We complete the topic with some discussions in Chapter 5.

Graphs. Following [3, 41], we consider graphs without loops or multiple edges (Actually, our results generalize to graphs with loops and multiple edges, see Chapter 5). Considering the property of the edge coloring we are dealing with, we may also assume without loss of generality that the graphs under consideration do not have isolated vertices. A vertex u with degree 0 (1, respectively) is an *isolated vertex* (a *leaf-neighbor of* v, where $uv \in E(G)$, respectively). For a coloring φ, we say that an edge uv is *colored i, for some* $i \in [c]$, if $\varphi(u) = \varphi(v) = i$, otherwise we say that it is *uncolored*. Let $\varphi(X) = \{i \in [c] : \varphi(u) = i, u \in V\}$ be the set of colors used for coloring of X.

3.1.1 Bounding Number of Vertices in Kernel

In this section, we show that c-LOAD COLORING admits a kernel with less than $2ck$ vertices. The fact that $(ck-1)K_2$ is a No-instance suggests that this kernel bound is likely to be optimal. The kernelization can be carried out in time $O((cn)^2)$.

For any integer $i \geq 1$ and $\tau \in \{<, \leq, =, >, \geq\}$, $K_{1,\tau i}$ denotes a *star* $K_{1,j}$ such that $j\ \tau\ i$ and $j \geq 1$. For instance, $K_{1,\leq p}$ is a star with q edges, $q \in [p]$. Then, a $K_{1,\tau i}$*-graph* is a forest in which every component is a star $K_{1,\tau i}$, and a $K_{1,\tau i}$*-cover* of G is a spanning subgraph of G which is a $K_{1,\tau i}$-graph. We call any $K_{1,\tau i}$-graph a *star graph* and any $K_{1,\tau i}$-cover a *star cover*.

We first prove the bound for star graphs with small maximum degree.

Lemma 3.1.1 *If G is a $K_{1,<2k}$-graph with $n \geq 2ck$, then $G \in (c,k)$-LC.*

Proof Let G be a $K_{1,<2k}$-graph with $n \geq 2ck$. We prove the lemma by induction on c. The base case of $c = 1$ holds since a $K_{1,<2k}$-graph has no isolated vertices: this property implies G has at least $\frac{|V(G)|}{2} \geq k$ edges.

Since all components of G are trees, for each one the number of vertices is one more than the number of edges. If there is a component C, with $k \leq |E(C)| < 2k$, we may color $V(C)$ with one color. Since we only used $|V(C)| \leq 2k$ vertices, $H = G - V(C)$ has at least $2(c-1)k$ vertices and so $H \in (c-1,k)$-LC by the induction hypothesis. Thus, $G \in (c,k)$-LC.

We may assume that every component has less than k edges and let $C_1, \ldots, C_t$ be the components of G. Let b be the minimum nonnegative integer for which there exists $I \subseteq [t]$ such that $\Sigma_{i \in I}|E(C_i)| = k + b \geq k$. Since there is no isolated vertex in a star graph, $m \geq \frac{n}{2} \geq ck$, and thus such a set I exists. Observe that for any $i \in I$, $|E(C_i)| > b$, as otherwise $\Sigma_{j \in I \setminus \{i\}}|E(C_j)| = k + b - |E(C_i)| \geq k$, a contradiction to the minimality of b. Since every component has less than k edges, $b \leq k-2$.

For a star (V,E), the ratio $\frac{|V|}{|E|}$ increases when $|E|$ decreases. Thus, we have $\Sigma_{j \in I}|V(C_j)| \leq \Sigma_{j \in I}|E(C_j)| \max_{h \in I}(\frac{|V(C_h)|}{|E(C_h)|}) \leq (k+b)\frac{b+2}{b+1}$. But $2k - (k+b)\frac{b+2}{b+1} = \frac{(k-2-b)b}{b+1} \geq 0$, and so $\Sigma_{j \in I}|V(C_j)| \leq 2k$. We may color the components C_i, $i \in I$, by the same color. Again, we have that $H = G - V(\bigcup_{i \in I} C_i)$ has at least $2(c-1)k$ vertices and so $H \in (c-1,k)$-LC by the induction hypothesis. Thus, $G \in (c,k)$-LC. ∎

For any star graph S and $\tau \in \{<, \leq, =, >, \geq\}$, let $C(S)$ ($L(S)$, respectively) be the centers (leaves, respectively) of stars in S (for the case of isolated edges in S, assign one vertex to $C(S)$ and one vertex to $L(S)$ arbitrarily). Let S_τ be the subgraph of S consisting of all stars whose centers v satisfy $d(v)\ \tau\ 2k-1$.

Corollary 3.1.1 *If $|C(S_{\geq})| + \frac{|V(S_{<})|}{2k} \geq c$, then $S \in (c,k)$-LC.*

Proof Clearly, $S_{\geq} \in (|C(S_{\geq})|, k)$-LC. We also have $S_{<} \in (\lfloor \frac{|V(S_{<})|}{2k} \rfloor, k)$-LC by Lemma 3.1.1. Thus, $S \in (|C(S_{\geq})| + \lfloor \frac{|V(S_{<})|}{2k} \rfloor, k)$-LC. ∎

We now introduce a family $(O_{i,k})_{i,k \in \mathbb{N}}$ of *overloads*.

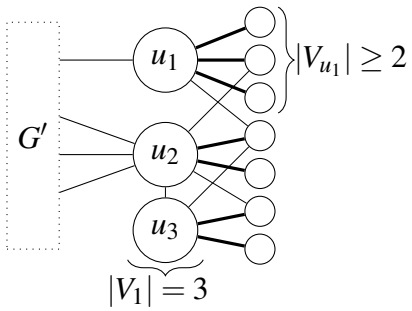

Fig. 3.1 An overload from $O_{3,2}$

Definition 3.1.1 *We call a pair (V_1, V_2) of disjoint vertex sets an* overload from $O_{i,k}$ *if $|V_1| = i$, $N(v) \subseteq V_1$ for all $v \in V_2$, and for every $u \in V_1$ there is a set $V_u \subseteq N_{V_2}(u)$ such that $|V_u| \geq k$ and for every pair u, v of distinct vertices of V_1, $V_u \cap V_v = \emptyset$ (see Fig. 3.1).*

Note that V_2 in Definition 3.1.1 is an independent set in the graph $G - V_1$.

If a graph G has an overload (V_1, V_2) from $O_{i,k}$, then $G[V_1 \cup V_2] \in (i,k)$-LC: for each $u \in V_1$, color $V_u \cup \{u\}$ with one color. From this observation, we deduce the following set of reduction rules:

Reduction rule $\mathbf{R_{i,k}}$. If an instance G of c-Load Coloring contains an overload $(V_1, V_2) \in O_{i,j}$, $j \geq k$, delete the vertices of $V_1 \cup V_2$ from G and decrease c by i.

Since the existence of an overload from $O_{i,j}$ for $i \geq c$ and $j \geq k$, in a graph G implies $G \in (c,k)$-LC, we only consider $\mathbf{R_{i,k}}$ for $i < c$. If it is not possible to apply any rule $\mathbf{R_{i,k}}$, $i < c$, to a graph G, we say that G is *irreducible for (c,k)-LC*, otherwise we apply the reduction rule and say that the resulting graph is *reduced from G using (V_1, V_2)*.

Observe that $\mathbf{R_{i,k}}$ may create isolated vertices, however, we will show in the following that we only use $\mathbf{R_{i,k}}$ in cases that do not produce isolated vertices.

Let G' be a graph reduced from G using $(V_1, V_2) \in O_{i,j}$, $i < c$.

Proposition 1 *If $G' \in (c-i,k)$-LC then $G \in (c, \min\{j,k\})$-LC.*

Proof Any (i,j)-coloring of the overload $(V_1,V_2) \in O_{i,j}$ is an $(i,min\{j,k\})$-coloring of it. Any $(c-i,k)$-coloring of G' of it. Merging the two coloring together, we obtain a $(c,\min\{j,k\})$-coloring of G. ∎

Proposition 2 *If $G \in (c,k)$-LC then $G' \in (c-i,k)$-LC.*

Proof Let us call an edge uncolored, if the color of its endvertices are different. In whatever (c,k)-coloring φ of G, any edge incident to V_1 is colored with a color in $\varphi(V_1)$ or is uncolored. Thus, there are at least $c-|V_1|$ colors for which all edges of that color are in $E(G-V_1)$. And by the definition of an overload, any vertex in V_2 is isolated in $G-V_1$. So, these colored edges are in $E(G-(V_1 \cup V_2)) = E(G')$. We conclude that $G' \in (c-|V_1|,k)$-LC. ∎

These two propositions imply that the reduction rules are safe.

Lemma 3.1.2 *Let G' be reduced from G using $(V_1,V_2) \in O_{i,j}$, $i < c$, $j \geq k$. Then $G \in (c,k)$-LC if and only if $G' \in (c-i,k)$-LC.*

We now describe our polynomial reduction algorithm.

Theorem 3.1.1 *Given two positive integers $c,k > 1$ and a graph G with $n \geq 2ck$ vertices, there exists an algorithm running in time $O((cn)^2)$ which decides $G \in (c,k)$-LC or outputs an instance (G',c') reduced from (G,c) using an overload from $O_{c-c',2k-1}$, where $c' \in [c-1]$, $|V(G')| < 2c'k$.*

Proof We first show that G has a star cover. Recall that we assume G has no isolated vertex. By choosing a spanning tree of each component of G, we obtain a forest F. If a tree in F is not a star, it has an edge between two non-leaves. As long as F contains such an edge, delete it from F. Observe that F becomes a star cover of G.

Let S be a star cover of G. If $S \in (c,k)$-LC, then $G \in (c,k)$-LC since S is a subgraph of G. So, if $|C(S_{\geq})| + \frac{|V(S_<)|}{2k} \geq c$, then the algorithm may decide $G \in (c,k)$-LC by Corollary 3.1.1. On the other hand, if $S_>$ is empty, $G \in (c,k)$-LC by Lemma 3.1.1. We may assume these two properties do not hold and thus $|C(S_{\geq})| \in [c-1]$.

From star cover S, we will try to find some overload (V_1,V_2) such that we can apply the reduction rule. Our main idea is to regard centers of "big" stars as candidates for V_1 and their leaves as candidates for V_2, in the hope of finding big stars whose leaves have no neighbors outside of V_1. If, unfortunately, the leaf has neighbor outside of V_1, we will improve the star cover, such that we decrease the number of vertices in $S_>$ and put more vertices into $S_{\leq}$, until we find an overload or we can conclude that the graph is a Yes-instance.

We will now show that we may modify the star cover S until one of the above properties holds or G contains an overload $(V_1,V_2) \in O_{c-c',2k-1}$. In particular, we will show that the

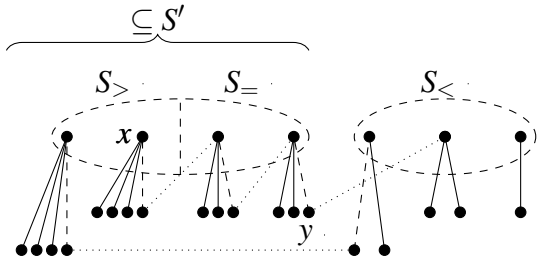

Fig. 3.2 $k = 2$, the dashed edges are in S, and will be deleted from it; the dotted edges are what we are looking for, and will be put into S.

modification can be done in time $O(c^2n)$ and it strictly decreases $|V(S_>)|$. Thus, the process may be applied at most n times and the resulting algorithm's running time is indeed $O((cn)^2)$.

We maintain a star graph S': initially, let $S' = S_>$ and **while** there is an edge $uv \in E(G) \setminus E(S)$ such that $u \in L(S')$ and $v \in C(S_= \setminus S')$, add the star centered at v to S'. Observe that this first construction runs in time $O(c^2n)$. Indeed, since $|C(S_\geq)| < c$, such an edge can be found in time $O(cn)$ and there are at most c steps in this while loop. As S' is a subgraph of the S, S' is a star graph.

Claim 1 *At any step of the construction of S' and for any leaf $y \in L(S')$, there exists an alternating path P from x to y such that $V(P) \subseteq S'$, $x \in C(S_>)$, the odd edges are in $E(S)$ and go from a center to a leaf, and the even edges are in $E(G) \setminus E(S)$ and go from a leaf to a center (see Fig. 3.2).*

We prove this claim by induction on the number of steps in the while loop. Initially, for any leaf y, the neighbor x of y is in $C(S_>)$, thus the desired path is $\{xy\}$. At any step, we add a vertex $v \in C(S_=)$ to S' because there exists an edge $uv \in E(G) \setminus E(S)$ such that u is a leaf introduced into S' before v. By induction hypothesis, there is a desired alternating path P from x to u such that $V(P) \cap N_S[v] = \emptyset$. Thus the desired alternating path for any leaf-neighbor y of v is $P \cup \{uv, vy\}$.

We say that we *reverse an alternating path from x to y in S* if we remove the odd edges from $E(S)$ and add the even edges into $E(S)$. This operation decreases the size of the star centered at x by 1, does not change the size of the transitional stars and isolates y. Since the length of a path is bounded by $2|C(S')| < 2c$, we may save these paths during the construction of S', and thus a reversal costs constant time.

Now we show how to handle the remaining problematic edges, i.e. edges $uv \in E(G) \setminus E(S)$ such that $u \in L(S')$ and $v \in V(G) \setminus C(S')$ (see Fig. 3.2). Recall that $v \notin C(S_\geq)$ by the construction of S' and there is an alternating path P from a vertex $x \in C(S_>)$ to u by Claim

1. In any of the following cases, we show how to modify S such that $|V(S_>)|$ decreases (by reversing a path) and such that the resulting graph remains a star cover :

- $v \in C(S_<)$ or v is the leaf of a single-leaf star in S : we reverse P in S and add uv to $E(S)$. Despite the reversal, the vertex u is not isolated in the resulting graph because of uv and v does not become the center of a star of size greater than $2k-1$.
- $v \in L(S)$ and v is not the leaf of a single-leaf star in S. Let y be the neighbor of v in S.
  - $vy \notin E(P)$: we reverse P in S, add uv to $E(S)$ and remove vy from it. Observe that the vertices u and v do not become isolated. The vertex y is not isolated either. Indeed, if $x \neq y$, y loses only one neighbor but it was not the center of a single-leaf star in S, and otherwise, it loses two neighbors but since $y = x \in C(S_>)$ and $k > 1$, $d_S(y) - 2 > 2k-3 \geq 1$.
  - $P = P' \cup \{yv\} \cup P''$: let w be the neighbor of u in S, we reverse $P' \cup \{yv\}$ in S, add vu and remove uw from $E(S)$. Again, u and v do not become isolated because of uv. Also, as $w \neq x$ (since they are different vertices of a path), w loses only one neighbor. We have $d_S(w) - 1 = 2k-2 \geq 2$ and w does not become isolated.

So, we may assume there is no edge $uv \in E(G) \setminus E(S)$ such that $u \in L(S')$ and $v \in V(G) \setminus C(S')$. Then any vertex in $L(S')$ is isolated in $G - C(S')$. Thus, for any $u \in L(S')$, we have $N(u) \subseteq C(S')$, and for each $v \in C(S')$, we can define V_v to be the leaves of the star centered at v, for which we have $|V_v| \geq 2k-1$. These two observations imply $(C(S'), L(S'))$ is an overload from $O_{|C(S')|,2k-1}$. Since the reductions are safe, the algorithm may output $(G', c') = (G - V(S'), c - |C(S')|)$. Note that $|V(G')| = 2k|C(S_= \setminus S')| + |V(S_<)| < 2k(c - |C(S')|) = 2c'k$ by the first assumption of the second paragraph of this proof. Since $C(S_>) \subseteq C(S') \subseteq C(S_\geq)$, we have $|C(S')| \in [c-1]$ (by the second paragraph of this proof) and therefore $c' = c - |C(S')| \in [c-1]$. Observe moreover that the reduced graph G' contains the star cover $S - V(S')$ and thus has no isolated vertices.

We finally discuss how to find such an edge in at most $O(c^2 n)$ time if it exists. We may assume we initially computed the degree of each vertex of G once (in time $O(n^2)$) and we can make copies of this information in time $O(n)$. Then, we may compute the degree of each vertex of the graph $G - C(S')$ in $O(cn)$ time since $|C(S')| < c$. We only need to know if there is a vertex $u \in L(S')$ such that $d_{V \setminus C(S')}(u) > 0$. If so, u is not isolated in $G - C(S')$ and ∎

Theorem 3.1.2 *For any fixed $c \geq 2$ and for any positive integer k, c-Load Coloring admits a kernel with less than $2ck$ vertices.*

Proof Observe first that $G \in (c,1)$-LC if and only if G has a matching with at least c edges. Since this property can be decided in polynomial time, we just need to consider the case when $k > 1$ and the input G has at least $n \geq 2ck$ vertices. Thus, the algorithm of Theorem 3.1.1 may decide whether $G \in (c,k)$-LC or obtains an instance (G',c') reduced from (G,c) such that $|V(G')| < 2c'k$. ∎

3.1.2 Bounding Number of Edges in Kernel

Let $S(c)$ be the integer sequence defined by induction by $S(1) = 1$, $S(2c) = 4S(c)$ and $S(2c+1) = 2S(c) + 2S(c+1)$. This sequence is known as A073121 in the Online Encyclopedia of Integer Sequences [88] (see also [5]). We will use the following technical result.

Proposition 3 *If c is even, $S(c) \leq \frac{9c^2-4}{8}$, and for any c, $S(c) \leq \frac{9c^2-1}{8}$.*

Proof It is easy to check the base cases: $S(1) = 1 = \frac{9(1)^2-1}{8}$, $S(2) = 4 = \frac{9(2)^2-4}{8}$ and $S(3) = 10 = \frac{9(3)^2-1}{8}$. We now assume the claim holds for every $c \leq 2c' - 1$ and we will prove it for $c = 2c'$ and $c = 2c' + 1$.
For even value, we have:

$$S(2c) = 4S(c) \leq 4\frac{9c^2-1}{8} = \frac{9(2c)^2-4}{8}.$$

For odd value, we have:

$$\begin{aligned} S(2c+1) &= 2(S(c) + S(c+1)) \\ &\leq 2\frac{9c^2 + 9(c+1)^2 - 1 - 4}{8} = \frac{9(2c+1)^2-1}{8}. \end{aligned}$$

∎

By using the kernel in the previous section, we show that c-LOAD COLORING admits a kernel with less than $(2S(c) + 4c^2 - 5c)k$ edges. Because of the upper bound on $S(c)$ given by Proposition 3, the number of edges in a kernel may be bounded by $6.25c^2k$. We first prove a smaller bound for bipartite graphs.

Lemma 3.1.3 *Let $b(c,k,n) = S(c)k + (c-1)n$. For every positive integer c and bipartite graph G with n vertices, if $m \geq b(c,k,n)$ then $G \in (c,k)$-LC.*

Proof We prove the lemma by induction on c. For the base case, observe that any graph with at least $k = b(1,k,n)$ edges is in $(1,k)$-LC for every k and n. We now assume the claim holds for every $c \leq 2c' - 1$ and we will prove it for $c = 2c'$ and $c = 2c' + 1$.

Suppose that $G = (A \cup B, E)$ is a bipartite graph with n vertices and at least $b(c,k,n)$ edges, but $G \notin (c,k)$-LC. Let B_2 be a maximal subset of B such that

$$|E(A,B_2)| < b(c-c',k,|A|+|B_2|)+b(c-c',k,|B_2|) \tag{3.1}$$

So, for any vertex $u \in B \setminus B_2$, the set $B_2 \cup \{u\}$ does not satisfy (3.1). Such a set B_2 exists since the empty set satisfies (3.1). Moreover, for any partition (A_1,A_2) of A, we know there exists $i \in \{1,2\}$ such that

$$|E(A_i,B_2 \cup \{u\})| \geq b(c-c',k,|A_i|+|B_2 \cup \{u\}|) \tag{3.2}$$

as otherwise, the linearity in n of $b(c,k,n)$ implies a contradiction with the maximality of B_2:

$$\begin{aligned} |E(A,B_2 \cup \{u\})| &= |E(A_1,B_2 \cup \{u\})| + |E(A_2,B_2 \cup \{u\})| \\ &< b(c-c',k,|A_1|+|B_2 \cup \{u\}|)+b(c-c',k,|A_2|+|B_2 \cup \{u\}|) \\ &= b(c-c',k,|A|+|B_2 \cup \{u\}|)+b(c-c',k,|B_2 \cup \{u\}|). \end{aligned}$$

Let $B_1 = B \setminus B_2$, $A_1 = A$ and $A_2 = \emptyset$. We define the following inequalities.

$$|E(A_1,B_1)| < b(c',k,|A_1|+|B_1|)+|A_1| \tag{3.3}$$

$$|E(A_2,B_1)| < b(c',k,|A_2|+|B_1|)+|A_2|. \tag{3.4}$$

While (3.3) does not hold and (3.4) holds, we move an arbitrary vertex from A_1 to A_2. Suppose eventually (3.3) and (3.4) are both false and let u be an arbitrary vertex in B_1. We deduce for both $i = 1$ and $i = 2$ that

$$|E(A_i,B_1 \setminus \{u\})| \geq b(c',k,|A_i|+|B_1|).$$

Thus, there exist disjoint vertex sets X and Y such that $|E(X)| \geq b(c',k,|X|)$ and $|E(Y)| \geq b(c-c',k,|Y|)$ (either $X = A_1 \cup B_1 \setminus \{u\}$ and $Y = A_2 \cup B_2 \cup \{u\}$, or $X = A_2 \cup B_1 \setminus \{u\}$ and $Y = A_1 \cup B_2 \cup \{u\}$, depending on whether (3.2) holds for $i = 1$ or $i = 2$). By taking a (c',k)-coloring of X and a $(c-c',k)$-coloring of Y, we have that $G \in (c,k)$-LC, a contradiction.

So, we may assume (3.3) eventually holds. If $A_2 = \emptyset$, then $|E(A_2, B_1)| = 0$. Otherwise, let v be the last vertex moved from A_1 to A_2. Observe that

$$\begin{aligned} |E(A_2,B_1)| &\leq |E(A_2 \setminus \{v\}, B_1)| + |B_1| \\ &< b(c', k, |A_2 \setminus \{v\}| + |B_1|) + |A_2 \setminus \{v\}| + |B_1| \text{ (by (3.4)).} \\ &< b(c', k, |A_2| + |B_1|) + |A_2| + |B_1|. \end{aligned} \tag{3.5}$$

In both cases, (3.5) holds and we can bound the number of edges in G:

$$\begin{aligned} |E(G)| &= |E(A,B_2)| + |E(A_1,B_1)| + |E(A_2,B_1)| \\ &< b(c-c', k, |A| + |B_2|) + b(c-c', k, |B_2|) \\ &+ b(c', k, |A_1| + |B_1|) + |A_1| \\ &+ b(c', k, |A_2| + |B_1|) + |A_2| + |B_1| \\ &\quad \text{(by inequalities (3.1),(3.3),(3.5)).} \end{aligned}$$

If $c = 2c'$, we have $c - c' = c'$ and

$$|E(G)| < 4S(c')k + 2(c'-1)n + n = b(c,k,n).$$

Otherwise, $c = 2c' + 1$ and then $c - c' = c' + 1$. Thus,

$$\begin{aligned} |E(G)| &< 2S(c')k + 2S(c'+1)k + 2(c'-1)n \\ &\quad + |A| + 2|B_2| + |A_1| + |A_2| + |B_1| \\ &\leq S(2c'+1)k + 2c'n = b(c,k,n). \end{aligned}$$

Thus, for $c = 2c'$ and $c = 2c' + 1$, we have $|E(G)| < b(c,k,n)$, a contradiction. So, there is no bipartite graph with n vertices and at least $b(c,k,n)$ edges such that $G \notin (c,k)$-LC. ∎

We now generalize this lemma for any graph. We would like to find a partition (A,B) of V such that $|E(A)| + |E(B)|$ is bounded, since $|E(A,B)|$ is bounded.

Lemma 3.1.4 *Let $f(c,k,n) = (2S(c) - c)k + 2(c-1)n$. For every positive integer c and every graph G with n vertices, if $m \geq f(c,k,n)$ then $G \in (c,k)$-LC.*

Proof We prove the lemma by induction on c. For the base case, observe that any graph with at least $k = f(1,k,n)$ edges is in $(1,k)$-LC for every k and n. We now assume the claim holds for every $c \leq 2c' - 1$ and we will prove it for $c = 2c'$ and $c = 2c' + 1$.

Consider a graph G with n vertices and at least $f(c',k,n)$ edges, such that $G \notin (c,k)$-LC. We will first show that there exists a set $A \subseteq V(G)$ such that $f(c',k,|A|) \leq |E(A)| \leq$

$f(c',k,|A|)+|A|$ (and thus $G[A] \in (c',k)$-LC). We may construct the set A as follows: initially $A = \emptyset$ and while $|E(A)| < f(c',k,|A|)$, add an arbitrary vertex of $V(G) \setminus A$ to A. Let u be the last added vertex. Then

$$|E(A)| \leq |E(A \setminus \{u\})| + |A \setminus \{u\}| < f(c',k,|A \setminus \{u\}|) + |A \setminus \{u\}| < f(c',k,|A|) + |A|.$$

Let $B = V(G) \setminus A$. If $G[B] \in (c-c',k)$-LC, then $G \in (c,k)$-LC, a contradiction. So $|E(B)| < f(c-c',k,|B|)$. Furthermore, $|E(A,B)| < b(c,k,n)$ by Lemma 3.1.3. Finally, we may bound $|E(G)|$. If $c = 2c'$, we have $c - c' = c'$

$$\begin{aligned} |E(G)| &< f(c',k,|A|) + f(c',k,|B|) + b(2c',k,n) + |A| \\ &= (2S(c') - c')k + 2(c'-1)|A| + (2S(c') - c')k + 2(c'-1)|B| \\ &\quad + S(2c')k + (2c'-1)n + |A| \\ &\leq (2S(2c') - 2c')k + (4c'-2)n = f(c,k,n). \end{aligned}$$

Otherwise, $c = 2c'+1$ and $c - c' = c'+1$. Thus,

$$\begin{aligned} |E(G)| &< f(c',k,|A|) + f(c'+1,k,|B|) + b(2c'+1,k,n) + |A| \\ &= (2S(c') - c')k + 2(c'-1)|A| + (2S(c'+1) - (c'+1))k + 2c'|B| \\ &\quad + S(2c'+1)k + 2c'n + |A| \\ &\leq (2S(2c'+1) - (2c'+1))k + 4c'n = f(c,k,n). \end{aligned}$$

Thus, in both cases $|E(G)| < f(c,k,n)$, as required. ∎

Recall that Proposition 3 implies $f(c,k,2ck) < 6.25c^2k$. Thus Lemma 3.1.4 implies the following

Corollary 3.1.2 *For every graph G with less than $2ck$ vertices, if $m \geq 6.25c^2k$ then $G \in (c,k)$-LC.*

Theorem 3.1.2 and Corollary 3.1.2 imply the following.

Theorem 3.1.3 *The c-LOAD COLORING Problem admits a kernel with less than $f(c,k,2ck) < 6.25c^2k$ edges.*

The size of this kernel may be optimal up to a constant factor. Indeed, the complete bipartite graph $K_{c,ck-1}$ is an irreducible graph for (c,k)-LC with $c^2k - c = O(c^2k)$ edges, but $K_{c,ck-1} \notin (c,k)$-LC. We can increase this lower bound by joining all c vertices on the smaller side of $K_{c,ck-1}$. The resulting graph is not in (c,k)-LC either, and it has $c^2k + \frac{c(c-3)}{2}$ edges.

3.1.3 Approximation Algorithm

We consider an approximation algorithm for the MAX c-LOAD COLORING problem. Given a graph G and integer c, we wish to determine $k_{opt}(G,c)$, the maximum k for which $G \in (c,k)$-LC. Given an approximation algorithm, we define the approximation ratio as $\frac{k_{opt}(G,c)}{k}$, where k is the output of the approximation algorithm.

Note that $k_{opt}(G,c) \leq \lfloor \frac{|E(G)|}{c} \rfloor$ by the pigeonhole principle. Let $K(c)k$ be an upper bound of the number of edges in a kernel for c-LOAD COLORING. By Theorem 3.1.3, we may have $K(c) = 6.25c^2$.

Theorem 3.1.4 *Given a graph G and a positive integer c, there exists an algorithm running in time $O(c^3n^2)$ which outputs k such that $G \in (c,k)$-LC and $\frac{k_{opt}(G,c)}{k+1} < \frac{K(c)}{c} = 6.25c$.*

Proof We prove the claim by induction on c. If $c = 1$, the algorithm trivially outputs $|E(G)|$. We assume the claim holds for any $i < c$, and want to prove it for c.

Let $k = \lfloor \frac{|E(G)|}{K(c)} \rfloor$. Note that $k+1 > \frac{|E(G)|}{K(c)} \geq \frac{ck_{opt}(G,c)}{K(c)}$, thus $\frac{K(c)}{c} > \frac{k_{opt}(G,c)}{k+1}$. We also have $\frac{K(c)}{c} > \frac{k_{opt}(G,c)}{2k-1}$ if $k > 1$.

If $k \leq 1$, since $G \in (c,1)$-LC if and only if G has a matching with at least c edges, the algorithm may decide whether $G \in (c,1)$-LC in time $O(c^2n)$ using any matching algorithm. Depending on the answer, the algorithm outputs $k_{opt}(G,c) = k = 0$ or $k = 1$. Therefore we may assume $k > 1$.

If $n < 2ck$, and as we choose k such that $m \geq K(c)k$, Corollary 3.1.2 implies $G \in (c,k)$-LC. Thus, the algorithm may output k. Otherwise, we may give G as input of Theorem 3.1.1's algorithm for c-LOAD COLORING. Again, if the answer is $G \in (c,k)$-LC, our approximation algorithm may output k. Otherwise the algorithm of Theorem 3.1.1 returns a graph G' reduced from G using an overload from $O_{c-c',2k-1}$, where $c' \in [c-1]$.

So now assume we have such a G'. Since $G \in (c,k_{opt}(G,c))$-LC, we have $G' \in (c',k_{opt}(G,c))$-LC by Proposition 2. Thus $k_{opt}(G,c) \leq k_{opt}(G',c')$ and by induction hypothesis, we may find an integer k' such that $G' \in (c',k')$-LC and $6.25c > 6.25c' > \frac{k_{opt}(G',c')}{k'+1} \geq \frac{k_{opt}(G,c)}{k'+1}$. As we also have $G \in (c,\min\{2k-1,k'\})$-LC by Proposition 1, let the algorithm output $\min\{2k-1,k'\}$.

The time complexity of the algorithm follows from the complexity of the algorithm of Theorem 3.1.1 and the fact that any step of the induction needs to use the reduction algorithm only once and this strictly decreases c. ∎

Note that Theorem 3.1.4 does not technically give us a $\frac{K(c)}{c} = 6.25c$ approximation, as we only have $\frac{k_{opt}}{k+1} < \frac{K(c)}{c}$ rather than $\frac{k_{opt}}{k} < \frac{K(c)}{c}$. However, the following holds:

Corollary 3.1.3 *There is a $12.5c$ approximation algorithm for* MAX c-LOAD COLORING.

Proof By construction, if the approximation algorithm outputs $k = 0$, then $k_{opt}(G,c) = 0$. Otherwise, $k \geq 1$ and then $\frac{k_{opt}}{k} \leq \frac{2k_{opt}}{k+1} < 2\frac{K(c)}{c} = 12.5c$. ∎

3.1.4 Number of Edges in Kernel for $c = 2$

In this section, we look into the edge kernel problem for the special case when $c = 2$. By doing a refined analysis, we will give a kernel with less than $6k + (3+\sqrt{2})\sqrt{k}$+4 edges for 2-LOAD COLORING, which is a better bound than the general one.

Lemma 3.1.5 *If a graph G is irreducible for $(2,k)$-LC and $\Delta(G) \geq 3k$, then $G \in (2,k)$-LC.*

Proof Let u be one of the vertices with degree Δ and $N(u)$ its neighbors. Since G is reduced by Reduction Rule $\mathbf{R_{1,k}}$, u has at least $2k$ neighbors which are not leaves. Thus, these vertices are incident to at least k edges not incident with u. Arbitrarily color k of them with color 1. By construction, there are at most $2k$ colored vertices. So there are at least $\Delta - 2k \geq k$ uncolored vertices in $N(u)$. We color them and u with 2. Thus, $G \in (2,k)$-LC. ∎

We first establish a bound of the number of edges in a particular kind of minimal vertex subsets.

Lemma 3.1.6 *Let k be a positive integer. For any $V' \subseteq V(G)$ such that $|E(V')| = k + d \geq k$ and V' contains at most one vertex u with $d_{V'}(u) \leq d$, we have $d < \sqrt{2k}$.*

Proof Since $|E(V')| \geq k > 0$, V' has at least two vertices, and thus there exists a vertex $v \in V'$ such that $d < d_{V'}(v) \leq |V'| - 1$. This implies

$$2(k+d) = 2|E(V')| = \sum_{v \in V'} d_{V'}(v) \geq (d+1)(|V'|-1) \geq (d+1)^2.$$

Thus, $2k \geq d^2 + 1$ implying $d < \sqrt{2k}$. ∎

The following lemmas and corollaries bound sizes of some sets of edges in a partition of $V(G)$ in three sets.

Lemma 3.1.7 *Let G have a partition $V(G) = A \cup B_1 \cup B_2$ and let $s = \min_{i \in [2]} |B_i|$. If $|E(A,B_i)| + 2|E(B_i)| \geq 2k + s$ for $i \in [2]$, then $G \in (2,k)$-LC.*

Proof For any 2-coloring of G, any $i \in [2]$ and any disjoint vertex sets X, Y, we denote by $E_i(X)$ ($E_i(X,Y)$, respectively) the set of edges colored i from $E(X)$ (E(X,Y), respectively). Throughout the proof, all vertices of B_i, $i \in [2]$, will be colored i, and therefore $E_i(B_i) = E(B_i)$.

Let $B = B_1 \cup B_2$, and for each $B' \subseteq B$, let $A[B'] = \{u \in A : N_B(u) = B'\}$. Also, let $A_i[B']$ be the set of vertices from $A[B']$ colored i.

Let us color vertices of A as follows.

If there is a set $B' = B'_1 \cup B'_2$, such that $B'_1 \subseteq B_1$, $B'_2 \subseteq B_2$, and $|A[B']|$ is even, then we assign half of the vertices of $A[B']$ color 1, and the other half color 2. We have

$$|E_i(A[B'], B_i)| = |A_i[B']|\ |B'_i| = \frac{|A[B']|}{2}\ |B'_i| \text{ for both } i \in [2].$$

If there are two sets $B' = B'_1 \cup B'_2$ and $B'' = B''_1 \cup B''_2$, such that $|A[B']|$ and $|A[B'']|$ are odd, $B'_1, B''_1 \subseteq B_1$, $B'_2, B''_2 \subseteq B_2$, and $|B'_1| \geq |B''_1|$, $|B'_2| \leq |B''_2|$, then assign $\frac{|A[B']|+1}{2}$ vertices of $A[B']$ and $\frac{|A[B'']|-1}{2}$ vertices of $A[B'']$ color 1, and $\frac{|A[B']|-1}{2}$ vertices of $A[B']$ and $\frac{|A[B'']|+1}{2}$ vertices of $A[B'']$ color 2. We have

$$\begin{aligned} |E_1(A[B'], B_1)| + |E_1(A[B''], B_1)| &= \frac{(|A[B']|+1)}{2}|B'_1| + \frac{(|A[B'']|-1)}{2}|B''_1| \\ &\geq \frac{|A[B']|}{2}|B'_1| + \frac{|A[B'']|}{2}|B''_1|. \end{aligned}$$

and, similarly, $|E_2(A[B'] \cup A[B''], B_2)| \geq \frac{|A[B']|}{2}|B'_2| + \frac{|A[B'']|}{2}|B''_2|$.

Let us denote by $B^1, B^2, \ldots, B^t$ the remaining subsets for which $A[B^j]$ is uncolored, and for $i \in [2]$ and $j \in [t]$, define $B^j_i = B^j \cap B_i$. Since for every pair of uncolored sets B^j, B^h, we have that either $|B^j_1| > |B^h_1|$ and $|B^j_2| > |B^h_2|$, or $|B^j_1| < |B^h_1|$ and $|B^j_2| < |B^h_2|$, we may order these sets such that for all j, h, $0 < j < h \leq t$, we have $|B^j_1| > |B^h_1|$ and $|B^j_2| > |B^h_2|$. Without loss of generality, let us assume $|B_1| \leq |B_2|$.

For each $j \in [t]$, assign $\frac{|A[B^j]|+1}{2}$ vertices of $A[B^j]$ color 1 if j is even, and assign $\frac{|A[B^j]|-1}{2}$ vertices of $A[B_j]$ color 1, otherwise. Assign the remaining vertices of $A[B^j]$ color 2. Then we have that

$$\begin{aligned} \sum_{j=1}^{t} |E_2(A[B^j], B_2)| &= \sum_{j=1}^{t} \left(\frac{|A[B^j]|}{2}|B^j_2|\right) + \sum_{j \text{ odd}} \frac{|B^j_2|}{2} - \sum_{j \text{ even}} \frac{|B^j_2|}{2} \\ &\geq \sum_{j=1}^{t} \left(\frac{|A[B^j]|}{2}|B^j_1|\right) + \sum_{j=1}^{\lfloor t/2 \rfloor} \frac{|B^{2j-1}_1| - |B^{2j}_1|}{2} \\ &\geq \sum_{j=1}^{t} \left(\frac{|A[B^j]|}{2}|B^j_1|\right) 2. \end{aligned}$$

We also have that

$$\begin{aligned}\sum_{j=1}^{t}|E_1(A[B^j],B_1)| &= \sum_{j=1}^{t}(\frac{|A[B^j]|}{2}|B_1^j|) - \sum_{j\text{ odd}}\frac{|B_1^j|}{2} + \sum_{j\text{ even}}\frac{|B_1^j|}{2}\\ &\geq \sum_{j=1}^{t}(\frac{|A[B^j]|}{2}|B_1^j|) - \frac{|B_1^1|}{2} + \sum_{j=1}^{\lfloor(t-1)/2\rfloor}\frac{|B_1^{2j}|-|B_1^{2j+1}|}{2}\\ &\geq \sum_{j=1}^{t}(\frac{|A[B^j]|}{2}|B_1^j|) - \frac{s}{2}.\end{aligned}$$

Observe that we have colored all the vertices. Since all the sets $A[B']$ are disjoint, we may sum up all the inequalities we have so far for both $i\in[2]$ to obtain:

$$\begin{aligned}2|E_i(V(G))| &\geq 2|E_i(B_i)|+2|E_i(A,B_i)|\\ &= 2|E(B_i)|+2\sum_{B'\subseteq B}|E_i(A[B'],B_i)|\\ &\geq 2|E(B_i)|+\sum_{B'\subseteq B}|A[B']|\,|B_i'|-s\\ &= 2|E(B_i)|+|E(A,B_i)|-s\geq 2k,\end{aligned}$$

which means $|E_i(V(G))|\geq k$ for both $i\in[2]$, and so $G\in(2,k)$-LC. ∎

Corollary 3.1.4 *Let k and s be two positive integers and $V_> = \{u\in V(G) : d(u)\geq\frac{2k}{s}+s+1\}$. If $|V_>|\geq 2s$ then $G\in(2,k)$-LC.*

Proof Let B_1 and B_2 two disjoint arbitrary subsets of $V_>$ such that $|B_1|=|B_2|=s$ and let $A=V(G)\setminus(B_1\cup B_2)$. Observe that for both $i\in[2]$, every vertex $u\in B_i$ has at most $|B_{3-i}|=s$ neighbors in $|B_{3-i}|$, thus $d_{A\cup B_i}(u)=d_{V(G)\setminus B_{3-i}}(u)\geq\frac{2k}{s}+1$. We deduce

$$|E(A,B_i)|+2|E(B_i)| = \sum_{u\in B_i}d_{A\cup B_i}(u)\geq|B_i|(\frac{2k}{s}+1)=2k+s.$$

So, by Lemma 3.1.7, $G\in(2,k)$-LC. ∎

Corollary 3.1.5 *Let k be a positive integer and $V_> = \{u\in V(G) : d(u)\geq 3\sqrt{k}+4\}$. If $|V_>|\geq 2\lfloor\sqrt{k}\rfloor$ then $G\in(2,k)$-LC.*

Proof Let $s=\lfloor\sqrt{k}\rfloor\geq 1$ and $e=\sqrt{k}-s<1$. Corollary 3.1.4 applies since

$$\frac{2k}{s}+s+1=\frac{2(s+e)^2}{s}+s+1=3s+4e+\frac{2e^2}{s}+1=3\sqrt{k}+e+\frac{2e^2}{s}+1<3\sqrt{k}+4.$$

∎

If $|B_1| = 1$, we may obtain a better result than Lemma 3.1.7.

Lemma 3.1.8 *Let G have a partition $V(G) = A \cup \{u\} \cup B_2$ such that $d_A(u) \geq 2k$. If $|E(A,B_2)| + |E(B_2)| \geq k$, then $G \in (2,k)$-LC.*

Proof Choose a minimal set $A' \subseteq A$, such that $|E(A',B_2)| + |E(B_2)| \geq k$. Observe that for all $v \in A'$, $d_{B_2}(v) \geq 1$, as otherwise, such a vertex would not contribute to $|E(A',B_2)| + |E(B_2)|$ and we may delete it, a contradiction with the minimality of A'. Then we have $|A'| \leq k$, and thus $d_{A \setminus A'}(u) \geq d_A(u) - |A'| \geq k$. We may color $A' \cup B_2$ with one color and $(A \setminus A') \cup \{u\}$ with another, which implies that $G \in (2,k)$-LC. ∎

Corollary 3.1.6 *Let $G \notin (2,k)$-LC. Then there exists at most one vertex such that $d(v) > 2k$.*

Proof Suppose there are at least two vertices u and v with degree greater than $2k$ and let $A = V(G) \setminus \{u,v\}$. We have $d_A(u) \geq 2k$ and $d_A(v) \geq 2k$, then Lemma 3.1.8 applies and $G \in (2,k)$-LC, a contradiction. ∎

With these observations, we may prove the main lemma of this section.

Lemma 3.1.9 *Let $\Delta(G) < 3k$ and $|E(G)| \geq 6k + (3+\sqrt{2})\sqrt{k} + 4$. Then $G \in (2,k)$-LC.*

Proof Let G be a graph with at least $6k + (3+\sqrt{2})\sqrt{k} + 4$ edges and $\Delta = \Delta(G) < 3k$, but $G \notin (2,k)$-LC. Let $t = 3\sqrt{k} + 4$, $V_> = \{x \in V(G) : d(x) \geq t\}$ and $A = V(G) \setminus V_>$. Let u be a vertex of degree Δ. By Corollary 3.1.5, for every $v \in V(G) \setminus \{u\}$, $d(v) \leq 2k$.

As $t = \frac{2k}{\sqrt{k}} + \sqrt{k}$, we have $|V_>| < 2\sqrt{k}$, as otherwise, $G \in (2,k)$-LCP by Corollary 3.1.4, a contradiction. Thus, for any partition X, Y of $V_>$ we have

$$|E(X,Y)| \leq |X||Y| < |X|(2\sqrt{k} - |X|) \leq k \tag{3.6}$$

We will now show that there exists a partition $V(G) = A' \cup B_1 \cup B_2$ such that $|E(A')| = k + d < k + \sqrt{2k}$, $|E(B_1 \cup B_2)| < k$ and $|E(A',B_i)| \geq 2k$, for both $i \in [2]$. Let us consider the following two cases.

Case 1: $d_A(u) \geq 2k$. Let $B_1 = \{u\}$ and $B_2 = V_> \setminus B_1$. By Lemma 3.1.8, we have $|E(A,B_2)| + |E(B_2)| < k$. If $|E(A)| < k$ then $|E(G)| = |E(A)| + |E(A,B_2)| + |E(B_2)| + d(u) < k + k + \Delta < 5k$, a contradiction. Thus, we may assume $|E(A)| \geq k$. We take a minimal set $A' \subseteq A$, such that $k \leq |E(A')| = k + d$. Observe that if there is any vertex $v \in A'$ with $|N'_A(v)| \leq d$, then

$A'' = A' \setminus \{v\}$ is a smaller vertex set such that $|E(A'')| \geq k$, a contradiction to the minimality of A'. Thus, by Lemma 3.1.6 we have $d < \sqrt{2k}$. Let $B' = A \setminus A'$. Note that we have $|E(B_1 \cup B_2 \cup B')| < k$ as otherwise $G \in (2,k)$-LC since $|E(A')| \geq k$.

Suppose $|E(A', \{u\})| \geq 2k$. Then $|E(A', B_2 \cup B')| + |E(B_2 \cup B')| \geq k$ implies $G \in (2,k)$-LC by Lemma 3.1.8. So we have $|E(G)| = |E(A')| + |E(A', B_2 \cup B')| + |E(B_2 \cup B')| + d(u) < k + d + k + \Delta < 5k + d$, a contradiction.

So, we may assume $|E(A', B_1)| < 2k$. While this inequality holds, push a vertex from B' to B_1. Observe that after any move $|E(A', B_1)| < 2k + t$ since $\max\{d(v) : v \in B'\} \leq t$. Suppose that B' is empty but $|E(A', B_1)| < 2k$. Then $|E(A', B_2)| \leq |E(A, B_2)| + |E(B_2)| < k$ by Lemma 3.1.8, and $|E(G)| = |E(A')| + |E(A', B_1)| + |E(A', B_2)| + |E(B_1 \cup B_2)| < k + d + 2k + k + k = 5k + d$, a contradiction. Thus, $|E(A', B_1)| \geq 2k$ and we will put the remaining vertices of B' to B_2. We also have $|E(A', B_2)| \geq 2k$ as otherwise $|E(G)| = |E(A')| + |E(A', B_1 \cup B_2)| + |E(B_1 \cup B_2)| < (k + d) + (4k + t) + k = 6k + d + t$, a contradiction. So, we have the desired partition.

Case 2: $d_A(u) < 2k$. Recall that $|V_>| \leq 2\sqrt{k}$. Choose first a maximal set $B_1 \subseteq V_>$, such that $u \in B_1$, $|E(A, B_1)| + 2|E(B_1)| < 2k + \sqrt{k}$. Then choose a maximal set $B_2 \subseteq V_> \setminus B_1$, such that $|E(A, B_2)| + 2|E(B_2)| < 2k + \sqrt{k}$. Let $R = V_> \setminus (B_1 \cup B_2)$. If $|R| \geq 2$, put one vertex in B_1 and one in B_2, and then for both $i \in [2]$, the maximality of B_i implies $|E(A, B_i)| + 2|E(B_i)| \geq 2k + \sqrt{k}$, and so $G \in (2,k)$-LC by Lemma 3.1.7, a contradiction.

Thus, we may assume that R is empty or has one vertex. If R is not empty, let $R = \{r\}$ and recall that $d(r) \leq 2k$. Suppose that $|E(A \cup R)| < k$. By (3.6), $|E(B_1, B_2)| < k$. Thus, $|E(G)| = |E(A \cup R)| + |E(A, B_1)| + |E(B_1)| + |E(A, B_2)| + |E(B_2)| + |E(B_1, B_2)| + |E(R, B_1 \cup B_2)| < k + 4k + 2\sqrt{k} + k + |B_1 \cup B_2| < 6k + 4\sqrt{k}$, a contradiction.

Thus, we may assume that $|E(A \cup R)| \geq k$. Let A' be a minimal subset of $A \cup R$ such that $R \subseteq A'$ and $k \leq |E(A')| = k + d$. There is no $v \in A' \setminus R$ with $|N'_A(v)| \leq d$, and so by Lemma 3.1.6 we have $d < \sqrt{2k}$. Let $B' = A \setminus A'$ and observe that $|E(B_1 \cup B_2 \cup B')| < k$ as otherwise $G \in (2,k)$-LC since $|E(A')| \geq k$.

If $|E(A', B_1)| \geq 2k$, we still have $|E(A', B_1)| \leq |E(A, B_1)| < 2k + \sqrt{k} < 2k + t$. Otherwise, while $|E(A', B_1)| < 2k$ holds, push a vertex from B' to B_1. Observe that after any move $|E(A', B_1)| < 2k + t$. In any case, $|E(A', B_1)| < 2k + t$. Suppose that B' becomes empty while $|E(A', B_1)| < 2k$. Then $|E(A', B_2)| \leq |E(A, B_2)| + 2|E(B_2)| + |E(R, B_2)| < 2k + \sqrt{k} + 2\sqrt{k} = 2k + t$ and we have the bound $|E(G)| = |E(A')| + |E(A', B_1)| + |E(A', B_2)| + |E(B_1 \cup B_2)| < (k + d) + 2k + (2k + t) + k = 6k + t + d$, a contradiction. So $|E(A', B_1)| \geq 2k$ and we move the remaining vertices of B' to B_2. Suppose $|E(A', B_2)| < 2k$, we also have the bound $|E(G)| < 6k + t + d$, a contradiction. Thus, for both $i \in [2]$, we have $|E(A', B_i)| \geq 2k$. So, we

have the desired partition.

Let us consider such a partition. If there is a set $T \subseteq A'$, such that $|E(T,B_1)| > k$ and $|E(T,B_2)| \leq k$ (thus $|E(A' \setminus T,B_2)| \geq k$) or symmetrically, $|E(T,B_1)| \leq k$ (thus $E(A' \setminus T,B_1) \geq k$) and $|E(T,B_2)| > k$, then $G \in (2,k)$-LC, a contradiction. So, for any set $T \subseteq A'$, we have either $\max\{|E(T,B_1)|,|E(T,B_2)|\} \leq k$ or $\min\{|E(T,B_1)|,|E(T,B_2)|\} > k$. Select a maximal subset A_1 of A' such that $|E(A_1,B_i)| \leq k$ for $i \in [2]$. Observe that by construction in the two cases, A' contains at most one vertex r such that $d(r) > t$, and such a vertex has $d(r) \leq 2k$. In the construction of A_1, we may assume that r is the first element added to A_1 (note that $|E(\{r\},B_i)| \leq k$ for $i \in [2]$, as otherwise $|E(\{r\},B_i)| > k$ for $i \in [2]$ and $d(r) > 2k$, a contradiction). Thus, we may assume that $d(v) \leq t$ for every $v \in A' \setminus A_1$. Let $A_2 = A' \setminus (A_1 \cup \{v\})$, where v is an arbitrary vertex in $A' \setminus A_1$, and observe that $|E(A_2,B_i)| < k$ for $i \in [2]$ or $G \in (2,k)$-LC. The partition $A' = A_1 \cup A_2 \cup \{v\}$ satisfies $\max\{|E(A_i,B_j)| : i,j \in [2]\} \leq k$. Thus, $|E(A',B_1 \cup B_2)| < 4k + d_{B_1 \cup B_2}(v) \leq 4k + t$, and so $|E(G)| < 6k + t + d$, a contradiction. ∎

Lemmas 3.1.5 and 3.1.9 imply the following:

Theorem 3.1.5 *If G is irreducible for $(2,k)$-LC and has at least $6k + (3+\sqrt{2})\sqrt{k} + 4$ edges, then $G \in (2,k)$-LC. Thus,* 2-LOAD COLORING *admits a kernel with less than $6k + (3+\sqrt{2})\sqrt{k} + 4$ edges.*

3.2 Linear Kernel for Star Packing on Graphs with No Long Induced Paths

In this section, we will consider H-PACKING when $H = K_{1,r}$ and study $K_{1,r}$-PACKING from parameterized preprocessing, i.e., kernelization, point of view.

$K_{1,r}$-PACKING
Instance: A connected graph $G = (V,E)$, two positive integers k and r.
Parameter: k.
Output: Decide if there are k vertex disjoint copies of $K_{1,r}$ in G.

As a parameterized problem, $K_{1,r}$-PACKING was first considered by Prieto and Sloper [80], who obtained an $O(k^2)$-vertex kernel for each $r \geq 2$ and a kernel with at most $15k$ vertices for $r = 2$. Since the case $r = 1$, which is equivalent to computing the maximum matching, is polynomial-time solvable, we may restrict the discussion to $r \geq 2$. Fellows *et al.* [33] proved the same result for $r = 2$ and it later was improved to $7k$ by Wang *et al.* [104].

Fellows *et al.* [33] point out that, using their approach, the bound of [80] on the number of vertices in a kernel for any $r \geq 3$ can likely be improved to subquadratic. We believe that, in fact, there is a linear-vertex kernel for every $r \geq 3$, and we prove it on graphs with no long induced paths to support our conjecture. A path P in a graph G, is called *induced* if it is an induced subgraph of G. For an integer $d \geq 3$, let $\mathscr{G}_d$ denote the set of all graphs with no induced path on d vertices.

Theorem 3.2.1 *Let $r \geq 2$ and $d \geq 3$ be given integers. Then the $K_{1,r}$-PACKING problem restricted to graphs in $\mathscr{G}_d$ admits a kernel with $O(k)$ vertices.*

As d can be any arbitrary integer larger than two, $\mathscr{G}_d$ contains an ever increasing class of graphs which, in the "limit", coincides with all graphs. To show that Theorem 3.2.1 is an optimal[1] result, in a sense, we prove that $K_{1,r}$-PACKING restricted to graphs in $\mathscr{G}_d$ is NP-hard already for $d = 5$ and every fixed $r \geq 3$:

Theorem 3.2.2 *Let $r \geq 3$. It is NP-hard to decide if the vertex set of a graph in $\mathscr{G}_5$ can be partitioned into vertex-disjoint copies of $K_{1,r}$.*

We cannot replace $\mathscr{G}_5$ by $\mathscr{G}_4$ (unless $NP = P$) due to the following assertion.

Theorem 3.2.3 *Given graph $G \in \mathscr{G}_4$ and integer $r \geq 3$. We can compute the maximum number of vertex-disjoint copies of $K_{1,r}$ in G in polynomial time.*

[1] If $K_{1,r}$-PACKING was polynomial time solvable, then it would have a kernel with $O(1)$ vertices.

We give some terminology and notation exclusively for section 3.2 here. We say a star *intersects* a vertex set if the star uses a vertex in the set. We use (G,k,r) to denote an instance of the r-star packing problem. If there are k vertex-disjoint r-stars in G, we say (G,k,r) is a YES-instance, and we write $G \in \star(k,r)$. Given disjoint vertex sets S,T and integers s,r, we say that S has *s r-stars in T* if there are s copies of vertex-disjoint r-stars with centers in S and leaves in T.

3.2.1 Proof of Theorem 3.2.1

Note that the 1-star packing problem is the classic maximum matching problem and if $k = 1$, the r-star packing problem is equivalent to deciding whether the maximum degree in G $\Delta(G) \geq r$. Both of these problems can be solved in polynomial time. Henceforth, we assume $r > 1$ and $k > 1$.

A vertex u is called a *small vertex* if $\max\{d(v) : v \in N[u]\} < r$. A graph without a small vertex is a *simplified graph*.

We now give two reduction rules for an instance (G,k,r) of $K_{1,r}$-PACKING.

Reduction Rule 1 *If graph G contains a small vertex v, then return the instance $(G-v,k,r)$.*

It is easy to observe that Reduction Rule 1 can be applied in polynomial time.

Reduction Rule 2 *Let $G = (V,E)$ be a graph and let C,L be two vertex-disjoint subsets of V. The pair (C,L) is called a constellation if $G[C \cup L] \in \star(|C|,r)$ and every vertex in L is small in the graph $G[V \setminus C]$. If (C,L) is a constellation, return the instance $(G[V \setminus (C \cup L)], k-|C|)$.*

It is easy to observe that Reduction Rule 2 can be applied in polynomial time, provided we are given a suitable constellation. Also note that in our definition of a constellation, we allow $C = L = \emptyset$. In this case, we call it a trivial constellation, otherwise, we say it is a non-trivial constellation.

Lemma 3.2.1 *Reduction Rules 1 and 2 are safe.*

Proof Clearly, there is no r-star that uses the small vertex v. Therefore Reduction Rule 1 is safe as there are the same number of r-stars in G and $G-v$.

To see that Reduction Rule 2 is safe, it is sufficient to show that $G \in \star(k,r)$ if and only if $G[V \setminus (C \cup L)] \in \star(k-|C|,r)$. On the one hand, $G[V \setminus (C \cup L)] \in \star(k-|C|,r)$ together with the hypothesis $G[C \cup L] \in \star(|C|,r)$ implies $G \in \star(k,r)$. On the other hand, there are at most $|C|$ vertex-disjoint stars intersecting C. But by the definition of a constellation, every star intersecting L also intersects C. We know that there are at most $|C|$ stars intersecting $C \cup L$,

and so if $G \in \star(k,r)$, there are at least $k-|C|$ stars inside $G[V-(C\cup L)]$: $G[V\setminus(C\cup L)] \in \star(k-|C|,r)$. ∎

Note that as both reduction rules modify a graph by deleting vertices, any graph G' that is derived from a graph $G \in \mathscr{G}_d$ by an application of Reduction Rules 1 or 2 is still in $\mathscr{G}_d$.

Recall the Expansion Lemma, which is a generalization of the well-known Hall's theorem.

Lemma 3.2.2 *(**Expansion Lemma**)[32] Let r be a positive integer, and let m be the size of the maximum matching in a bipartite graph G with vertex bipartition $X\cup Y$. If $|Y| > rm$, and there are no isolated vertices in Y, then there exist nonempty vertex sets $S\subseteq X, T\subseteq Y$ such that S has $|S|$ r-stars in T and no vertex in T has a neighbor outside S. Furthermore, the sets S,T can be found in polynomial time in the size of G.*

We now provide a modified version of the Expansion Lemma. We will make use of it in our kernelization algorithm.

Lemma 3.2.3 *(**Modified Expansion Lemma**) Let r be a positive integer and let G be a bipartite graph with vertex bipartition $X\cup Y$. If $|Y| > r|X|$ and there are no isolated vertices in Y, then there exists a polynomial algorithm(in the size of G) which returns a partition $X = A_1\cup B_1$, $Y = A_2\cup B_2$, such that B_1 has $|B_1|$ r-stars in B_2, $E(A_1,B_2)=\emptyset$, and $|A_2| \le r|A_1|$.*

Proof If $|Y| \le r|X|$, then we may return $A_1 = X$, $A_2 = Y$, $B_1 = B_2 = \emptyset$. Otherwise, apply the Expansion Lemma to get nonempty vertex sets $S\subseteq X, T\subseteq Y$ such that S has $|S|$ r-stars in T and no vertex in T has a neighbor in Y outside S. Let $X' = X\setminus S$ and $Y' = Y\setminus T$. If $G[X'\cup Y']$ has isolated vertices in Y', move all of them from Y' to T. If $|Y'| \le r|X'|$, we may return $A_1 = X'$, $A_2 = Y'$, $B_1 = S$, and $B_2 = T$.

So now assume $|Y'| > r|X'|$. In this case, apply the algorithm recursively on $G[X'\cup Y']$ to get a partition $X' = A'_1\cup B'_1, Y' = A'_2\cup B'_2$, such that B'_1 has $|B'_1|$ stars in B'_2, $E(A'_1,B'_2)=\emptyset$, and $|A'_2| \le r|A'_1|$. Then return $A_1 = A'_1$, $B_1 = B'_1\cup S$, $A_2 = A'_2$, $B_2 = B'_2\cup T$. Observe that B_1 has $|B'_1|+|S| = |B_1|$ stars in B_2, $E(A_1,B_2)\subseteq E(A'_1,B'_2)\cup E(X\setminus S,T)=\emptyset$, and $|A_2| = |A'_2| \le r|A'_1| = r|A_1|$, as required. As each iteration reduces $|X|$ by at least 1, we will have to apply less than $|X|+|Y|$ iterations, each of which uses at most one application of the Expansion Lemma, and so the algorithm runs in polynomial time. ∎

Proof of Theorem 3.2.1. By exhaustively applying Reduction Rule 1, we may assume we have a simplified graph. Let G be a simplified graph in $\mathscr{G}_d$. Now find a maximal r-star packing of G with q stars. We may assume $q < k$ as otherwise we have a trivial YES-instance. Let S be the set of vertices in this r-star packing, and let $D = V(G)\setminus S$.

For any $u \in D$, let $D[u]$ be the set of vertices $v \in D$ for which there is a path from v to u using only vertices in D - that is, $D[u]$ is the the set of vertices in the component of $G[D]$ containing u. As our star-packing is maximal, $d_D(v) < r$ for every $v \in D$. As $G \in \mathscr{G}_d$, every $v \in D[u]$ has a path to u in $G[D]$ with at most $d-1$ vertices (as otherwise the shortest path in $G[D]$ from v to u is an induced path on at least d vertices). It follows that $|D[u]| \leq 1 + r + r^2 + \cdots + r^{d-1} \leq r^d$. It means that each component in $G[D]$ has bounded number of vertices, and we also have $|S| \leq (r+1)(k-1)$. So to bound the number of vertices in G, we just need to bound the number of components in $G[D]$.

Recursively, we put all vertices in S that has less than r edges to D into a set $Small(S)$, and put all components adjacent to such vertices into $B(D)$, the size bounded part in D. Following this idea, we show how to find a partition of S into $Big(S) \cup Small(S)$, and D into $B(D) \cup U(D)$, such that $|B(D)| \leq r^{d+1}|Small(S)|$, and $|U(D) \cap N(Big(S))| \leq r|Big(S)|$. Note that in our partition, $(Big(S), U(D))$ is a constellation. If $(Big(S), U(D))$ is a trivial constellation, i.e. $Big(S) = U(D) = \emptyset$, then as $|Small(S)| \leq |S| \leq (r+1)k$, it follows that $|V(G)| \leq (r+1)k + (r+1)r^{d+1}k$. Otherwise we can apply Reduction Rule 2 on $(Big(S), U(D))$.

We will construct $Big(S), Small(S), B(D), U(D)$ algorithmically in a way described below. Throughout, we will preserve the following

Property 1: $|B(D)| \leq |Small(S)|r^{d+1}$,

Property 2: $U(D)$ has no neighbors in $Small(S) \cup B(D)$.

Set $Big(S) = S, U(D) = D, Small(S) = B(D) = \emptyset$ initially,.

While $|U(D) \cap N(Big(S))| > r|Big(S)|$, do the following steps.

If there is a vertex $u \in Big(S)$ such that $|N(u) \cap U(D)| < r$, let $X = \bigcup\{D[v] : v \in N(u) \cap U(D)\}$. Observe that as $|D[v]| \leq r^d$ for all $v \in D$, $|X| < r^{d+1}$. Now set $Small(S) = Small(S) \cup \{u\}$, $Big(S) = Big(S) \setminus \{u\}$, $B(D) = B(D) \cup X$, $U(D) = U(D) \setminus X$. It follows that Property 1 is preserved. Note that no vertex in the new $U(D)$ has a neighbor in X (as all neighbors of X in D lie in X). Similarly no vertex in the new $U(D)$ is adjacent to u (as such a vertex would be in the old $U(D)$ and so would have been added to X). Therefore there are still no edges between the new $U(D)$ and the new $Small(S) \cup B(D)$, and so Property 2 is also preserved.

Otherwise (if every vertex $u \in Big(S)$ has $|N(u) \cap U(D)| \geq r$), let H denote the maximal bipartite subgraph of G with vertex partition $Big(S) \cup (U(D) \cap N(Big(S))$, and apply the Modified Expansion Lemma to H. We will get a partition $Big(S) = A_1 \cup B_1$ and $U(D) \cap N(Big(S)) = A_2 \cup B_2$ such that $E(A_1, B_2) = \emptyset, |A_2| \leq r|A_1|$ and B_1 has $|B_1|$ r-stars in B_2.

If the Modified Expansion Lemma returns $A_1 = \emptyset$, then we claim that $(Big(S), U(D))$ is a non-trivial constellation. To see this, firstly note that $|Big(S)|$ has $|Big(S)|$ r-stars in $U(D)$. Secondly, note that since we chose the vertices of a maximal star packing for S, there is no

r-star contained in $G[U(D)]$. As $U(D)$ has no neighbors in $Small(S) \cup B(D)$, it follows that there is no r-star intersecting $U(D)$ in $G \setminus Big(S)$. Thus $(Big(S), U(D))$ is a constellation, in this case, apply Reduction Rule 2 using $(Big(S), U(D))$. This gives us a partition in which $Big(S) = U(D) = \emptyset$. Thus, we have that $|U(D) \cap N(Big(S))| \leq r|Big(S)|$.

So now assume that the Modified Expansion Lemma returns $Big(S) = A_1 \cup B_1$ with $A_1 \neq \emptyset$. Let $X = \bigcup\{D[v] : v \in N(A_1) \cap U(D)\}$. Note that as $E(A_1, B_2) = \emptyset$ and $|A_2| \leq r|A_1|$, we have $|X| \leq |\bigcup\{D[v] : v \in A_2\}| \leq |A_2|r^d \leq |A_1|r^{d+1}$. Then let $Small(S) = Small(S) \cup A_1$, $Big(S) = Big(S) \setminus A_1$, $B(D) = B(D) \cup X$, $U(D) = U(D) \setminus X$. Note that after this move, we still have that $|B(D)| \leq |Small(S)|r^{d+1}$, and $U(D)$ has no neighbors in $Small(S) \cup B(D)$. Observe that in this case, $|Big(S)|$ strictly decreases, so the algorithm must eventually terminate.

Thus we have found the partition of S into $Big(S) \cup Small(S)$, and D into $B(D) \cup U(D)$, such that $|B(D)| \leq r^{d+1}|Small(S)|$, and $|U(D) \cap N(Big(S))| \leq r|Big(S)|$. Note that every vertex $u \in U(D)$ is in $D[v]$ for some $v \in N(S)$ (as otherwise, either $\max\{d(v) : v \in N[u]\} < r$ or $G[D]$ contains an r-star, a contradiction in either case). Moreover such a v must be in $U(D) \cap N(Big(S))$, as there are no edges between $U(D)$ and $Small(S) \cup B(D)$. Thus $|U(D)| \leq r^d|U(D) \cap N(Big(S))| \leq r^{d+1}|Big(S)|$. Then we have $|V(G)| = |S| + |U(D)| + |B(D)| \leq |S| + r^{d+1}|Big(S)| + r^{d+1}|Small(S)| \leq (r^{d+1}+1)|S| \leq (k-1)(r+1)(r^{d+1}+1) = O(k)$.

3.2.2 Proof of Theorem 3.2.2

A *split graph* is a graph where the vertex set can be partitioned into a clique and an independent set.

An instance of the well-known NP-hard problem 3-DIMENSIONAL MATCHING contains a vertex set that can be partitioned into three equally large sets V_1, V_2, V_3 (also called partite sets). Let k denote the size of each of V_1, V_2, V_3. It furthermore contains a number of 3-sets containing exactly one vertex from each V_i, $i = 1, 2, 3$. The problem is to decide if there exists a set of k vertex disjoint 3-sets (which would then cover all vertices). Such a set of k vertex disjoint 3-sets is called a perfect matching. The 3-sets are also called *edges* (or *hyperedges*).

Theorem 3.2.4 *Let $r \geq 3$. It is NP-hard to decide if the vertex set of a split graph can be partitioned into vertex disjoint copies of $K_{1,r}$.*

Proof We will provide a reduction from 3-DIMENSIONAL MATCHING. Let $\mathscr{I}$ be an instance of 3-dimensional matching. Let V_1, V_2, V_3 denote the three partite sets of $\mathscr{I}$ and let E denote the set of edges in $\mathscr{I}$. Let $m = |E|$ and $k = |V_1| = |V_2| = |V_3|$. We will build a split graph $G_{\mathscr{I}}$ as follows. Let $V = V_1 \cup V_2 \cup V_3$ be the vertices of $\mathscr{I}$. Let X_1 be a set of m vertices and X_2 be a set of $m-k$ vertices and let $X = X_1 \cup X_2$. Let Y be a set of $(m-k)(r-1)$ vertices

and let W be a set of $k(r-3)$ vertices (if $r=3$ then W is empty). Let the vertex set of $G_{\mathscr{I}}$ be $V \cup X \cup Y \cup W$.

Add edges such that X becomes a clique in $G_{\mathscr{I}}$. Let each vertex in X_1 correspond to a distinct edge in E and connect that vertex with the 3 vertices in V which belongs to the corresponding edge in E. Furthermore add all edges from X_1 to W. Finally, for each vertex in X_2 add $r-1$ edges to Y in such a way that each vertex in Y ends up with degree one in $G_{\mathscr{I}}$. This completes the construction of $G_{\mathscr{I}}$.

Clearly $G_{\mathscr{I}}$ is a split graph as X is a clique and $V \cup Y \cup W$ is an independent set. We will now show that the vertex set of $G_{\mathscr{I}}$ can be partitioned into vertex disjoint copies of $K_{1,r}$ if and only if $\mathscr{I}$ has a perfect matching.

First assume that $\mathscr{I}$ has a perfect matching. Let $E' \subseteq E$ denote the edges of the perfect matching. For the vertices in X_1 that correspond to the edges in E' we include the three edges from each such vertex to V as well as $r-3$ edges to W. This can be done such that we obtain k vertex disjoint copies of $K_{1,r}$ covering all of V and W as well as k vertices from X_1. Now for each vertex in X_2 include the $r-1$ edges to Y as well as one edge to an unused vertex in X_1. This can be done such that we obtain an additional $m-k$ vertex disjoint copies of $K_{1,r}$. We have now constructed m vertex disjoint copies of $K_{1,r}$ which covers all the vertices in $G_{\mathscr{I}}$, as required.

Now assume that the vertex set of $G_{\mathscr{I}}$ can be partitioned into vertex disjoint copies of $K_{1,r}$. As $|V \cup W \cup Y \cup X| = m(r+1)$ we note that we have m vertex disjoint copies of $K_{1,r}$, which we will denote by $\mathscr{K}$. As all vertices in Y need to be included in such copies we note that every vertex of X_2 is the center vertex of a $K_{1,r}$. Let $\mathscr{K}'$ denote these $m-k$ copies of $K_{1,r}$. Each $K_{1,r}$ in $\mathscr{K}'$ must include 1 edge from X_2 to X_1. These $m-k$ edges form a matching, implying that $m-k$ vertices of X_1 also belong to the copies of $K_{1,r}$ in $\mathscr{K}'$. This leaves k vertices in X_1 that are uncovered and rk vertices in $V \cup W$ that are uncovered. Furthermore, as $V \cup W$ is an independent set, each copy of $K_{1,r}$ in $\mathscr{K} \setminus \mathscr{K}'$ must contain a vertex of X_1. As $|\mathscr{K} \setminus \mathscr{K}'| = k$ we note that the k copies of $K_{1,r}$ in $\mathscr{K} \setminus \mathscr{K}'$ must include exactly one vertex from X_1. Also as each vertex in X_1 has exactly three neighbours in V, each such $K_{1,r}$ also contains 3 vertices from V (as V needs to be covered) and therefore $r-3$ vertices form W. Therefore the k vertices in X_1 that belong to copies of $K_{1,r}$ in $\mathscr{K} \setminus \mathscr{K}'$ correspond to k edges in E which form a perfect matching in $G_{\mathscr{I}}$.

This completes the proof as we have shown that $G_{\mathscr{I}}$ can be partitioned into vertex disjoint copies of $K_{1,r}$ if and only if $\mathscr{I}$ has a perfect matching. ∎

The following lemma is known. We give the simple proof for completeness.

Lemma 3.2.4 *No split graph contains an induced path on 5 vertices.*

Proof Assume G is a split graph where $V(G)$ is partitioned into an independent set I and a clique C. For the sake of contradiction assume that $P = p_0p_1p_2p_3p_4$ is an induced P_5 in G. As I is independent we note that $\{p_0, p_1\} \cap C \neq \emptyset$ and $\{p_3, p_4\} \cap C \neq \emptyset$. As C is a clique there is therefore an edge from a vertex in $\{p_0, p_1\}$ to a vertex in $\{p_3, p_4\}$. This edge implies that P is not an induced P_5 in G, a contradiction. ∎

Proof of Theorem 3.2.2. By Lemma 3.2.4, $\mathscr{G}_5$ contains all split graphs. The result now follows immediately from Theorem 3.2.4. ∎

3.2.3 Proof of Theorem 3.2.3

Note that $\mathscr{G}_4$ is the family of *cographs* [19]. It is well-known [19] that any non-trivial (i.e., with at least two vertices) cograph G is either disconnected or its complement is disconnected. Below let n denote the order of G and let m denote the size of G. The following lemma is well-known.

Lemma 3.2.5 *For any graph G, we can in time $O(n^2)$ find the connected components of G and the connected components of the complement of G.*

Lemma 3.2.6 *For any $G \in \mathscr{G}_4$ and any $s \geq 1$ we can in time $O(n^2)$ find a set of s vertices, say S, in G such that $|N[S]|$ is maximum possible.*

Proof Let $C_1, C_2, \ldots, C_l$ be the connected components of G ($l \geq 1$). Assume first that all the components are non-trivial. As any induced subgraph of a cograph is also a cograph we note that the complement of each C_i is disconnected. Therefore for each $i = 1, 2, \ldots, l$ there exists a non-trivial (each part is non-empty) partition (X_i, Y_i) of $V(C_i)$ such that all edges exist between X_i and Y_i in G. Let m_i be maximum degree of a vertex in C_i for each $i = 1, 2, \ldots, l$.

The maximum number of vertices we can add to $N[S]$ by adding one vertex from C_i is $m_i + 1$ and the maximum number of vertices added to $N[S]$ by adding two vertices from C_i is $|V(C_i)|$ as we can add a vertex from X_i and one from Y_i. Therefore the maximum possible $|N[S]|$ is the sum of the s largest numbers in the set $m_1 + 1, m_2 + 1, \ldots, m_s + 1, (|V(C_1)| - m_1 - 1), (|V(C_2)| - m_2 - 1), \ldots, (|V(C_l)| - m_l - 1)$. Furthermore it is easy to find the actual set S.
It is not hard to modify the proof above for the case when some C_i's are trivial.

Now we are ready to prove the main result of this appendix.

Proof of Theorem 3.2.3: Let $G \in \mathscr{G}_4$ and let $r \geq 3$ be arbitrary. First assume that G is connected, which implies that the complement of G is disconnected. Let X and Y partition

$V(G)$ such that all edges exist between X and Y in G. We now consider two cases.

Case 1: $|X| > r|Y|$ or $|Y| > r|X|$. Without loss of generality, assume that $|X| > r|Y|$. In this case we recursively find the maximum number of r-stars we can pack into $G[X]$. Let m_x be the maximum number of r-stars in $G[X]$. If $(r+1)m_x + (r+1)|Y| \leq n$, then the optimal answer is that we can pack $m_x + |Y|$ r-stars into G as we can always find $|Y|$ r-stars with centers in Y and not touching the m_x r-stars we already found in $G[X]$. If $(r+1)m_x + (r+1)|Y| > n$, then the optimal solution is $\lfloor n/(r+1) \rfloor$ r-stars as we can pick $|Y|$ r-stars touching as few of the m_x r-stars in $G[X]$ as possible and then pick as many of the m_x r-stars that are left untouched. This completes this case.

Case 2: $|X| \leq r|Y|$ and $|Y| \leq r|X|$. Let $x = |X|$ and $y = |Y|$ and define a and b as follows:

$$a = \frac{ry - x}{r^2 - 1} \text{ and } b = \frac{rx - y}{r^2 - 1}$$

Let $a' = \lfloor a \rfloor = a - \varepsilon_a$ and $b' = \lfloor b \rfloor = b - \varepsilon_b$. We will first show that we can find $a' + b'$ r-stars such that a' of the r-stars have the center in X and all leaves in Y and b' of the r-stars have the center in Y and all leaves in X. This is possible due to the following:

$$\begin{aligned} a'r + b' &= (a - \varepsilon_a)r + (b - \varepsilon_b) \\ &= r\tfrac{ry-x}{r^2-1} + \tfrac{rx-y}{r^2-1} - (r\varepsilon_a + \varepsilon_b) \\ &= y - (r\varepsilon_a + \varepsilon_b) \end{aligned}$$

And, analogously,

$$b'r + a' = x - (r\varepsilon_b + \varepsilon_a)$$

As $0 \leq \varepsilon_a < 1$ and $0 \leq \varepsilon_b < 1$ we note that we cover all vertices in G except $r\varepsilon_a + \varepsilon_b + r\varepsilon_b + \varepsilon_a = (r+1)(\varepsilon_a + \varepsilon_b)$. Therefore the number of vertices we cannot cover by the r-stars above is strictly less than $2(r+1)$. If $(r+1)(\varepsilon_a + \varepsilon_b) < r+1$ then we have an optimal solution (covering all vertices except at most r), so assume that $(r+1)(\varepsilon_a + \varepsilon_b) \geq r+1$.

Clearly the optimal solution is either $a' + b'$ or $a' + b' + 1$. As we already have a solution with $a' + b'$ r-stars we will now determine if there is a solution with $a' + b' + 1$ r-stars.

If some vertex, say w_x, in X has degree at least r in $G[X]$, then there is indeed a solution with $a' + b' + 1$ r-stars, because of the following. As $(r+1)(\varepsilon_a + \varepsilon_b) \geq r+1$ we must have $\varepsilon_a > 0$ and $\varepsilon_b > 0$, which implies that we can pick an r-star with center in $w_x \in X$ and with at most $r\varepsilon_b + \varepsilon_a - 1$ leaves in X and at most $r\varepsilon_a + \varepsilon_b$ leaves in Y. Once this r-star has been picked it is not difficult to pick an additional a' r-stars with centers in X (and leaves in Y) and

b' r-stars with centers in Y (and leaves in X), due to the above. Therefore we may assume no vertex in X has degree at least r in $G[X]$. Analogously we may assume that no vertex in Y has degree at least r in $G[Y]$.

If there exists $a'+1$ vertices S_X in X such that $|N[S_X] \cap X| \geq a'+1+r-(r\varepsilon_a+\varepsilon_b)$, then proceed as follows. We can create $a'+1$ stars in $G[X]$ such that they together have exactly $r-(r\varepsilon_a+\varepsilon_b)$ non-centers. By the above each star has less than r leaves, so we can expand these $a'+1$ stars to r-stars by adding leaves from Y. This uses up $a'+1+r-(r\varepsilon_a+\varepsilon_b)$ vertices from X and $(a'+1)r-(r-(r\varepsilon_a+\varepsilon_b))$ vertices from Y. Adding an additional b' stars with the center in Y and all leaves in X uses up b' vertices from Y and rb' vertices from X. Therefore we have used $b'+a'r+(r\varepsilon_a+\varepsilon_b) = y$ vertices from Y and the following number of vertices from X,

$$a'+rb'+1+r-(r\varepsilon_a+\varepsilon_b) = x-(r\varepsilon_b+\varepsilon_a)+1+r-(r\varepsilon_a+\varepsilon_b) = x+1+r-(r+1)(\varepsilon_a+\varepsilon_b)$$

As $(r+1)(\varepsilon_a+\varepsilon_b) \geq r+1$ we note that we use at most x vertices from X and we have a solution with $a'+b'+1$ r-stars. Analogously if there exists $b'+1$ vertices S_Y in Y such that $|N[S_Y] \cap Y| \geq b'+1+r-(r\varepsilon_b+\varepsilon_a)$, we obtain $a'+b'+1$ r-stars. By applying Lemma 3.2.6 to $G[X]$ and $G[Y]$ we can decide the above in polynomial time.

We may therefore assume that no such S_X or S_Y exist. We will now show that $a'+b'$ is the optimal solution. For the sake of contradiction assume that we have a^* r-stars with centers in X and b^* r-stars with centers in Y, such that they are vertex disjoint and $a^*+b^* = a'+b'+1$. Without loss of generality we may assume that $a^* \geq a'+1$. The a^* r-stars with centers in X all have at least one leaf in Y as the maximum degree in $G[X]$ is less than r. Furthermore by the above (S_X does not exist) any $a'+1$ r-stars with centers in X have more than $r(a'+1)-(r-(r\varepsilon_a+\varepsilon_b))$ leaves in Y. Therefore we use strictly more than the following number of vertices in Y.

$$r(a'+1)-(r-(r\varepsilon_a+\varepsilon_b))+(a^*+b^*-(a'+1)) = ra'+r\varepsilon_a+\varepsilon_b+b' = y$$

This contradiction implies that the optimal solution is $a'+b'$ in this case. This completes the case when G is connected.

Finally assume that G is disconnected. In this case we recursively solve the problem for each connected component, which can be added together to get an optimal solution for G. It is not difficult to see that the above can be done in polynomial time. ∎

Chapter 4

Cycles and Acyclicity in edge colored graphs

4.1 Acyclicity in Edge-Colored Graphs

It is well-known and trivial to prove the fact that every undirected and directed graph with no cycles has no closed walks either. Surprisingly, it turns out that this is not the case for PC cycles and PC walks. In fact, the properties of having no PC cycles, having no PC closed trails, and having no PC closed walks, are all different from each other.

In this section, we introduce five types of PC acyclicity, in order to better understand the structure of PC acyclic edge-colored graphs. The different types of PC acyclicity are defined as follows.

Definition 4.1.1 *Let G be a c-edge-colored undirected graph with $c \geq 2$. An ordering $v_1, v_2, \ldots, v_n$ of vertices in G is of*

***Type 1**: if for every $i \in [n]$, all edges from v_i to all vertices in each connected component of $G[\{v_{i+1}, v_{i+2}, \ldots, v_n\}]$ have the same color;*

***Type 2**: if for every $i \in [n]$, all edges from v_i to $\{v_{i+1}, v_{i+2}, \ldots, v_n\}$ which are not bridges of $G[\{v_i, v_{i+1}, \ldots, v_n\}]$ have the same color;*

***Type 3**: if for every $i \in [n]$, all edges from v_i to $\{v_{i+1}, v_{i+2}, \ldots, v_n\}$ have the same color;*

***Type 4**: if for every $i \in [n]$, all edges from v_i to $\{v_{i+1}, v_{i+2}, \ldots, v_n\}$ have the same color and all edges from v_i to $\{v_1, v_2, \ldots, v_{i-1}\}$ have the same color;*

Type 5: *if for every $i \in [n]$, all edges from v_i to $\{v_{i+1}, v_{i+2}, \ldots, v_n\}$ have the same color and all edges from v_i to $\{v_1, v_2, \ldots, v_{i-1}\}$ have the same color but different from the color of edges from v_i to $\{v_{i+1}, v_{i+2}, \ldots, v_n\}$.*

Definition 4.1.2 *Let $i \in [5]$. G is* **PC acyclic of type *i*** *if it has an vertex ordering $v_1, v_2, \ldots, v_n$ of type i.*

It is easy to observe that by definition, the class of c-edge-colored acyclic graphs of type i contains the class of c-edge-colored acyclic graphs of type $i+1$, $i \in [4]$. We will show that the containments are proper in this chapter. We prove that graphs of the first two types are exactly those edge-colored graphs without PC cycles and without PC closed trails, respectively. These two classes of edge-colored graphs were well characterized by Yeo [109] and Abouelaoualim *et al.* [2], respectively. We will prove that graphs of PC acyclicity of type 3 are edge-colored graphs without PC walks. We do not have a "nice" characterization of edge-colored graphs of type 4. In fact, we show that it is NP-hard to recognize graphs of this type, which is somewhat surprising as we prove that recognition of all other types can be done in polynomial-time. We will prove that an edge-colored graph is acyclic of type 5 if and only if every vertex is incident to edges of at most two colors and every cycle C has a positive even number of vertices incident, in C, to edges of the same color. For 2-edge-colored graphs this is equivalent to being bipartite with no PC cycle. Therefore, for 2-edge-colored graphs, being PC acyclic of type 5 is equivalent to being bipartite and PC acyclic of type 1.

Using the five types of PC acyclicity, we further investigate the border between intractability and tractability for the problems of finding the maximum number of internally vertex-disjoint PC paths between two vertices and of deciding the minimum number of vertices to eliminate all PC paths between two vertices. We show that both problems are NP-hard for 2-edge-colored graphs of PC acyclicity of type 3 (and thus of types 1 and 2), but polynomial time solvable for edge-colored graphs of PC acyclicity of type 4 (and of type 5). We will also show that while Menger's theorem does not hold in general, even on 2-edge-colored graphs of PC acyclicity of type 3 (or 1 or 2), it holds on edge-colored graphs of PC acyclicity of type 4 (and 5).

The rest of the section is organized in the following way. In Section 4.1.1, we study the five types of acyclicity in edge-colored graphs. Section 4.1.2 is devoted to PC paths and separators in edge-colored graphs. We raise an open problem from FPT perspective in Chapter 5.

4.1.1 Types of PC Acyclicity Edge-Colored Graphs

In this section, we give good characterizations of the five types of PC acyclicity introduced in the previous section.

The fact that a c-edge-colored graph G is PC acyclic of type 1 if and only if G has no PC cycle follows immediately from a theorem by Yeo [109] below (a special case of Yeo's theorem for $c = 2$ was obtained by Grossman and Häggkvist [39]).

Theorem 4.1.1 *If a c-edge-colored graph G has no PC cycle then G has a vertex z such that every connected component of $G - z$ is joined to z by edges of the same color.*

Theorem 4.1.1 easily implies the following corollary.

Corollary 4.1.1 *A c-edge-colored graph G is PC acyclic of type 1 if and only if there is no PC cycle in G.*

Proof Let G be a c-edge-colored graph which is PC acyclic of type 1 and it has a PC cycle C. Just take an acyclic ordering θ of $V(G)$ of type 1. Let x be the vertex on C with lowest subscript in the acyclic ordering θ. Observe that $C - x$ must belong to the same component in $G - x$ and all vertices of $C - x$ have bigger subscript than x in θ. Thus, in the cycle C, the two incident edges of x must have the same color, a contradiction to the fact that C is PC.

Now suppose G has no PC cycle. By Theorem 4.1.1, there is a vertex z in G such that every connected component of $G - z$ is joined to z by edges of the same color. Set $v_1 = z$ and consider $G - z$ recursively to obtain $v_2, \ldots, v_n$. Clearly, the resulting ordering is PC acyclic of type 1. ∎

Theorem 4.1.1 implies that checking PC acyclicity of type 1 can be done in polynomial time. Indeed, to decide if G has no PC cycle, we just need to check if G has a vertex z such that every connected component of $G - z$ is joined to z by edges of the same color and $G - z$ has no PC cycle.

Using the next theorem, similarly to proving Corollary 4.1.1, we can show that a c-edge-colored graph G is PC acyclic of type 2 if and only if G has no PC closed trail.

Theorem 4.1.2 (Abouelaoualim *et al.* [2]) *Let G be a c-edge-colored graph, such that every vertex of G is incident with at least two edges of different colors. Then either G has a bridge or G has a PC closed trail.*

Theorem 4.1.2 implies that to check whether G contains a PC closed trail, it suffices to recursively delete all bridges and all G-monochromatic vertices. Observe that G has a PC

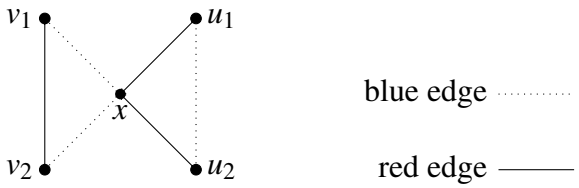

Fig. 4.1 Graph G of type 1 not type 2

trail if and only if the resulting graph is non-empty. This implies that checking PC acyclicity of type 2 can also be done in polynomial time.

To see that containment is proper between acyclicities of type 1 and type 2, consider a graph G (Fig 4.1) with vertex set $\{v_1, v_2, x, u_1, u_2\}$ and edge set $\{v_1v_2, v_1x, v_2x, u_1u_2, u_1x, u_2x\}$, where v_1v_2, u_1x, u_2x are colored red, u_1u_2, v_1x, v_2x are colored blue. Clearly, this 2-edge-colored graph G has no PC cycle, but it has a PC closed trail. Thus, G is acyclic of type 1 but not acyclic of type 2.

It seems that c-edge-colored graphs without PC closed walks have not been studied in the literature. Here is a counterpart of Theorem 4.1.1 for such graphs.

Theorem 4.1.3 *If a c-edge-colored graph G has no PC closed walk, then G has a G-monochromatic vertex.*

Proof For a given c-edge-colored graph G, we construct a c-edge-colored graph H which is called an *extension* of G. H is obtained from G by replacing every vertex u with a set I_u of independent vertices which have the same adjacencies and edge colors as u. Observe that G has no PC closed walk if and only if no extension of G, in which I_u is sufficiently large, has a PC cycle. Now apply Theorem 4.1.1 to an extension H of a connected c-edge-colored graph G in which $|I_u| > 1$ for each $u \in V(G)$, and note that for every vertex $z \in V(H)$, $H - z$ is connected. ∎

Using Theorem 4.1.3, similarly to proving Corollary 4.1.1, we can show that a c-edge-colored graph G is PC acyclic of type 3 if and only if G has no PC closed walk. Theorem 4.1.3 implies that G has no PC closed walk if and only if G has a vertex z incident with edges of the same color and $G - z$ has no PC closed walk. Thus, checking PC acyclicity of type 3 can be done in polynomial time.

To see that containment is proper between acyclicities of type 2 and type 3, consider the following graph G (Fig 4.2) with $V(G) = \{a_1, a_2, a_3, b_1, b_2, b_3\}$, with blue edges a_1b_1, a_2b_2 and a_3b_3 and red edges a_1a_2, b_1a_2, a_3b_2 and b_3b_2. In G, we have a PC closed walk $a_1a_2b_2b_3a_3b_2a_2b_1a_1$. This walk uses the edge a_2b_2 twice. There is no PC closed trail in G:

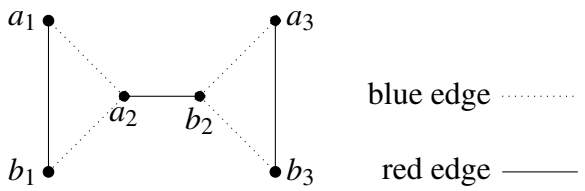

Fig. 4.2 Graph G of type 2 not type 3

as a_2b_2 is a bridge, it does not belong to a closed trail and removing a_2b_2 makes it obvious that there is no PC closed trail in the remainder.

There is unlikely a "nice" characterization of c-edge-colored graphs of type 4 due to the following somewhat surprising result.

Theorem 4.1.4 *It is NP-complete to decide whether a 2-edge-colored graph is acyclic of type 4.*

Proof It is easy to see that our problem is in NP, since given any vertex ordering of the graph, it is trivial to check whether it is acyclic of type 4.

To prove NP-hardness, we reduce from the following BETWEENNESS problem.

> BETWEENNESS
> *Instance:* A set of distinct ordered triples of elements from a universe U.
> *Output:* Decide whether there exists a linear ordering of elements in U, such that for every triple (x,y,z) of distinct elements of U in the input, either $x \succ y \succ z$ or $z \succ y \succ x$ in the ordering i.e., y must appear between x and z.

The BETWEENNESS problem was proved to be NP-complete [71].

In the rest of the proof, it will be convenient for us to write an ordering $v_1, v_2, \ldots, v_p$ of some vertices in a graph as $v_1 \succ v_2 \succ \cdots \succ v_p$.

Given an instance of BETWEENNESS, we produce a 2-edge-colored graph G as follows. We add each element in U as a vertex of G. (These vertices will all be incident only to blue edges in the final graph.) For each triple (x,y,z), we create a gadget with vertices x,y,z and new vertices $a(x,y), b(x,y), b(z,y), a(z,y)$. Add blue edges $xa(x,y)$, $b(x,y)b(z,y)$, $za(z,y)$, $yb(x,y)$ and $yb(z,y)$, and red edges $a(x,y)b(x,y)$, $b(z,y)a(z,y)$, see Figure 4.3. This concludes the construction of G.

Now suppose G is acyclic of type 4 and consider the ordering within the gadget for a triple (x,y,z). As $xa(x,y)b(x,y)b(z,y)a(z,y)z$ is a PC path, any acyclic ordering of type 4 must have either $x \succ a(x,y) \succ b(x,y) \succ b(z,y) \succ a(z,y) \succ z$ or $z \succ a(z,y) \succ b(z,y) \succ b(x,y) \succ a(x,y) \succ x$. Suppose the former. Then y cannot appear before $b(x,y)$ because of the red edge

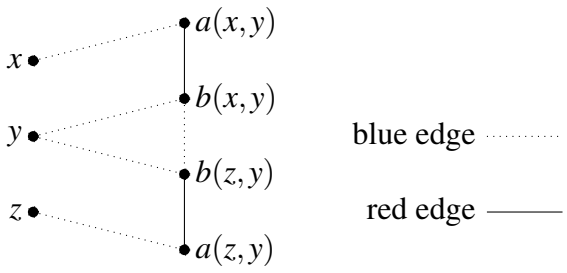

Fig. 4.3 (x,y,z)-gadget

$a(x,y)b(x,y)$, and it cannot appear after $b(z,y)$ because of the red edge $b(z,y)a(z,y)$. Thus y must appear between $b(x,y)$ and $b(z,y)$, and in particular y must appear between x and z. A similar argument holds when $z \succ a(z,y) \succ b(z,y) \succ b(x,y) \succ a(x,y) \succ x$.

Thus, if G is acyclic of type 4, then there is an ordering such that for every input triple (x,y,z), y appears between x and z, and so our Betweenness instance is a Yes-instance.

Conversely, suppose our Betweenness instance is a Yes-instance, consider an ordering of U satisfying every triple of the instance. We extend this to an ordering of $V(G)$. For each triple (x,y,z), if $x \succ y \succ z$, then we set $x \succ a(x,y) \succ b(x,y) \succ y \succ b(z,y) \succ a(z,y) \succ z$. If $z \succ y \succ x$, then we set $z \succ a(z,y) \succ b(z,y) \succ y \succ b(x,y) \succ a(x,y) \succ x$. As $a(x,y), b(x,y), b(z,y), a(z,y)$ are involved only in the (x,y,z)-gadget, there is an ordering of $V(G)$ that satisfies all the above orderings. It is easy to check that each of $a(x,y), b(x,y), b(z,y), a(z,y)$ is satisfied (in the sense of having all edges to earlier vertices the same color, and all edges to later vertices the same color). As all vertices from U are only incident to blue edges, they are satisfied by this ordering as well. Thus we have that G is acyclic of type 4, as required. ∎

To see that containment is proper between acyclicities of type 3 and type 4, consider a complete graph on three vertices with two blue edges and one red edge. It is easy to find a PC acyclic ordering of type 3 and to see that there is no PC acyclic ordering of type 4.

The following procedure provides an easy way to check whether a connected edge-colored graph G is acyclic of type 5.

Procedure 1 *First check that for each vertex in the graph, its incident edges are colored with at most two colors, as otherwise, G cannot be acyclic of type 5. Choose an arbitrary vertex x and orient edges of one color out of x and edges of the other color towards x. Then for every vertex y for which there is an arc towards (out of, respectively) y which was an edge of color i, mark y and orient all edges of color i incident to y towards (out of, respectively) y and all other edges incident to y out of y (towards y, respectively). Stop the procedure if orienting edges incident to y leads to one of the following conflicts for another vertex z:*

(a) *z will have two arcs of different color oriented into it or two arcs of different color oriented out of it;*

(b) *z will have an arc into it and an arc out of it of the same color.*

Theorem 4.1.5 will show that if this procedure is completed (without conflicts (a) and (b)) and the obtained digraph is acyclic then G is acyclic of type 5, and otherwise it is not. Note that Procedure 1 always finishes in polynomial time.

Here is a characterization of acyclic edge-colored graphs of type 5. Recall that a vertex v in an edge-colored graph G is G-monochromatic if all edges incident to v in G are of the same color.

Theorem 4.1.5 *Let G be an edge-colored graph. The following are equivalent.*

1. *G is PC acyclic of type 5.*
2. *Procedure 1 completes and the resulting digraph is acyclic.*
3. *Every vertex is incident to edges of at most two colors, and every cycle C in G has a positive even number of C-monochromatic vertices.*

Proof We will show that the implications $3 \Rightarrow 2 \Rightarrow 1 \Rightarrow 3$ hold.

First we will show that $2 \Rightarrow 1$. Assume that Procedure 1 completes and the resulting digraph, D, is acyclic. Let $v_1, v_2, v_3, \ldots, v_n$ be an acyclic ordering of D (that is, if $v_i v_j$ is an arc of D then $i < j$). By the construction of D, we note that all arcs into a vertex v_i have the same color and all arcs out of v_i also have the same color, which is different from the arcs entering v_i. As this holds for all v_i we note that the ordering $v_1, v_2, v_3, \ldots, v_n$ is a PC acyclic ordering of type 5 in G, which proves $2 \Rightarrow 1$.

We will now show that $1 \Rightarrow 3$, so assume that $\mathscr{O} = v_1, v_2, v_3, \ldots, v_n$ is a PC acyclic ordering of type 5 in G. By the definition of PC acyclic ordering of type 5, every vertex is incident to edges of at most two colors. Let C be any cycle in G. Let A_1 contain the vertices, v_i, of C where both neighbors of v_i on C lie after v_i in the ordering $\mathscr{O}$. Let A_2 contain the vertices, v_i, of C where both neighbors of v_i on C lie before v_i in the ordering $\mathscr{O}$. Let $B = V(C) \setminus (A_1 \cup A_2)$. That is, B contains the vertices, v_i, of C where one neighbors of v_i lies after v_i in the ordering $\mathscr{O}$ and the other lies before. Note that $|A_1| = |A_2|$, as the cycle changes from going "forward" to "backward" $|A_2|$ times and changes from going "backward" to "forward" $|A_1|$ times. Furthermore $|A_1| > 0$ (and $|A_2| > 0$) as the vertex of C with minimum index in $\mathscr{O}$ belongs to A_1. As the C-monochromatic vertices on C are exactly $A_1 \cup A_2$, we note that this is an even positive number ($= 2|A_1| = 2|A_2|$). This proves $1 \Rightarrow 3$.

We will now show that $3 \Rightarrow 2$. We will prove this by showing that if 2 is false, then 3 is false. So assume that Procedure 1 either does not complete or the resulting digraph, D, is not acyclic. First assume that it does complete, but the resulting digraph, D, is not acyclic and let C be a directed cycle in D. Note that in G there is no C-monochromatic vertices, which implies that 3 is false.

We may thus assume that Procedure 1 does not complete. Let $v_1, v_2, v_3, \ldots$ be the order in which the vertices are considered by the procedure. Let v_r be the first vertex (that is r is smallest possible) such that when orienting all edges incident to v_r, some other vertex, v_k, will have one of the two conflicts in Procedure 1:

(a) v_k will have two arcs of different color oriented into it or two arcs of different color oriented out of it;

(b) v_k will have an arc into it and an arc out of it of the same color.

Note that $k > r$, since otherwise we already considered all edges incident with v_k and when considering v_r, it will not orient any edges in an opposite direction to what it is already oriented (by the minimality of r). Note that when we consider some v_i, it has an arc to or from a vertex in $\{v_1, v_2, \ldots, v_{i-1}\}$ for all i. Therefore $G[v_1, v_2, \ldots, v_r]$ is connected. By (a) or (b), v_k has two edges that have already been oriented after considering the vertices $v_1, v_2, \ldots, v_r$ (and form a conflict as in (a) or (b)), namely $v_r v_k$ and $v_j v_k$ for some $j < r$. For a path P between v_r and v_j in $G[v_1, v_2, \ldots, v_r]$, let C be the cycle obtained by adding the edges $v_r v_k$ and $v_k v_j$ to P. Let A_1 be all vertices on C, where both edges in C are oriented into the vertex and let A_2 be all vertices on C, where both edges in C are oriented out the vertex. Note that all vertices in $V(C) \setminus \{v_k\}$ are C-monochromatic if and only if they belong to $A_1 \cup A_2$. Furthermore, v_k is C-monochromatic if and only if it does not belong to $A_1 \cup A_2$. Observe that $|A_1| = |A_2|$, so $|A_1 \cup A_2|$ is even, which implies that there are an odd number of C-monochromatic vertices on C, and so 3 does not hold. Therefore $3 \Rightarrow 2$. ∎

Theorem 4.1.5 implies the following simpler characterization for the case $c = 2$.

Corollary 4.1.2 *A 2-edge-colored graph G is PC acyclic of type 5 if and only if it is bipartite and has no PC cycle.*

Proof Observe that every cycle C in a 2-edge-colored graph has an even number of C-non-monochromatic vertices, since the edge color must change an even number of times as we go around the cycle back to the starting point. Therefore a 2-edge-colored graph contains an odd cycle if and only if it contains a cycle C with an odd number of C-monochromatic vertices. The result now follows by Theorem 4.1.5. ∎

To see that containment is proper between acyclicities of type 4 and type 5, consider any non-bipartite 2-edge-colored graph with all edges being blue.

4.1.2 PC Paths and Separators

This section will be devoted to separators and PC paths in edge-colored graphs. We consider two problems. Let a c-edge-colored graph G and distinct vertices $x, y \in V(G)$ be given.

MINIMUM PC SEPARATOR
Instance: An edge colored graph $G = (V, E)$, and two vertices $x, y \in V(G)$.
Output: A minimum-size set $S \subseteq V(G) \setminus \{x, y\}$ such that there is no PC path between x and y in $G - S$.

PC PATH PACKING
Instance: An edge colored graph $G = (V, E)$, and two vertices $x, y \in V(G)$.
Output: The maximum number of internally vertex-disjoint PC paths between x and y in G.

We will see that both the MINIMUM PC SEPARATOR problem and the PC PATH PACKING problem are NP-hard on graphs G that are PC acyclic of types 1, 2 or 3, even for $c = 2$. We also find that Menger's theorem fails to hold for these graphs.

On the other hand, we show that the analogue of Menger's theorem does hold for graphs G that are PC acyclic of type 4, and that both the minimum PC separator problem and the PC path packing problem are in P on these graphs, even if no acyclic ordering of type 4 is given. These results hold for arbitrary c.

We begin with the hardness results.

Theorem 4.1.6 *The minimum PC separator problem is NP-hard for 2-edge-colored graphs which are acyclic of type 3.*

Proof We give a reduction from the vertex cover problem which is to find a minimum size vertex cover of a given graph. Given an instance H of the vertex cover problem, we construct a 2-edge colored graph G which is PC acyclic of type 3, and two distinct vertices x, y, such that a set S of vertices is a vertex cover of H if and only if there is no PC path between x and y in $G - S$.

Let $V(G) = V(H) \cup \{x, y\}$ and $E(G) = E(H) \cup \{xu : u \in V(H)\} \cup \{vy : v \in V(H)\}$, and let us color all edges in $E(H)$ red and all the edges incident to x or y blue. It is easy to see that graph G is PC acyclic of type 3, just put x and y in the beginning of the vertex ordering.

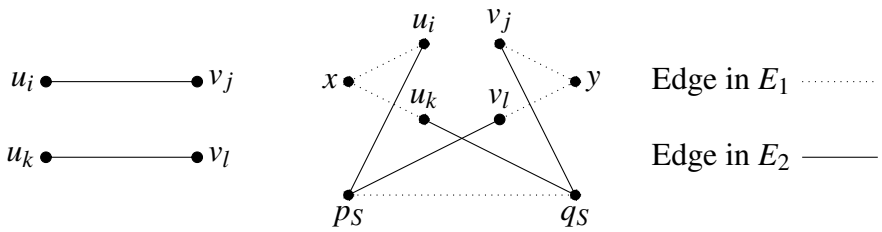

Fig. 4.4 Construction of H

Observe that for any vertex set $S \subseteq V(H) = V(G) \setminus \{x,y\}$, there is a PC path between x and y in graph $G-S$ if and only if there is at least one edge in the graph $H-S$. Thus a vertex set S is a vertex cover of H if and only if there is no PC path between x and y in $G-S$. Thus we have given a polynomial reduction from the vertex cover problem to our problem, which implies the NP-completeness. ∎

Theorem 4.1.7 *The PC path packing problem is NP-hard for 2-edge-colored graphs which are acyclic of type 3.*

Proof We will give a reduction from the following problem called the restricted bipartite perfect matching problem: Given a bipartite graph G and a partition of its edges into sets of size at most 2, decide whether G has a perfect matching containing at most one edge from each partition set (we will call such a perfect matching *restricted*). This problem was proved to be NP-complete by Plaisted and Zaks [78].

Let $(G,\mathscr{S})$ be an instance of the restricted bipartite perfect matching problem, where $G = (V_1 \cup V_2, E)$ is a bipartite graph with $|V_1| = |V_2|$, and $\mathscr{S}$ is the collection of partite sets of size 2. Note that we may require that for every size-2 set $\{e_i, e_j\} \in \mathscr{S}$ that e_i and e_j are vertex-disjoint; if not, we may split $\{e_i, e_j\}$ into two sets $\{e_i\}, \{e_j\}$ of size 1, as no matching can use both edges simultaneously.

We construct a PC acyclic 2-edge colored graph H of type 3 in the following way. Let E_1 and E_2 denote the sets of blue and red edges of H, respectively. Introduce two new vertices x,y, add all edges from $\{xu : u \in V_1\} \cup \{vy : v \in V_2\}$ to E_1. For each edge uv of E which is not in any 2-size set of S, we add uv to E_2. For any 2-size set $S = \{u_iv_j, u_kv_l\}$ of $\mathscr{S}$, where $\{u_i,u_k\} \subseteq V_1$, $\{v_j,v_l\} \subseteq V_2$, we add new vertices p_S, q_S to $V(H)$, add edges $u_ip_S, v_lp_S, u_kq_S, v_jq_S$ to E_2, and add edge p_Sq_S to E_1; see Figure 4.4. This completes the construction of H. Observe that any ordering of $V(H)$ starting with x and y then containing all vertices of G and finally having all other vertices, is PC acyclic of type 3.

Now we prove that there is a restricted perfect matching in G if and only if there are $|V_1|$ internally vertex-disjoint PC paths between x and y.

Firstly, if there is a restricted perfect matching in G, we can easily get $|V_1|$ internally vertex-disjoint alternating paths between x and y in H, by constructing a PC path for each edge in the perfect matching. For each edge u_iv_j in the perfect matching, forming a 2-size set S of $\mathscr{S}$ with some edge u_kv_l, we obtain a PC path $xu_ip_Sq_Sv_jy$ or $xu_iq_Sp_Sv_jy$ between x and y. For each other edge uv in the perfect matching, we have a PC path $xuvy$ between x and y.

Secondly, if there are $|V_1|$ internally vertex-disjoint PC paths in H between x and y, there must be a restricted perfect matching in G. Observe that all the PC paths are of length 3 or 5 as all vertices in $V(G)$ are adjacent to only color 2 edges in $H-\{x,y\}$. For each PC path we put an edge into the matching of graph G. Any PC path of length 3 must be of the form $xuvy$, so we put edge uv into the matching. For any edge p_Sq_S, there is at most one internally disjoint PC path passing through it, thus, if there is one such path $xu_ip_Sq_Sv_jy$ ($xu_kq_Sp_Sv_ly$, respectively), we put edge u_iv_j (u_kv_l, respectively) into the matching. Since we will put at most one edge from each set S of $\mathscr{S}$ into the matching, the matching is a restricted matching. Since there are $|V_1|$ internally vertex-disjoint PC paths between x and y, we will put $|V_1|$ non-adjacent edges into the matching, thus we obtain a perfect matching. ∎

The same hardness results also hold if we consider edge-disjoint PC paths instead of vertex-disjoint ones, by a simple reduction. Namely, let G be a 2-edge-colored graph. We split every vertex $v \in V(G)$ into two copies v' and v'', where v' is incident with all red edges incident with v, and v'' with all blue edges. We also add a third vertex v_0, a blue edge $v'v_0$ and a red edge v_0v''. It is easy to see that applying this transformation to every vertex in G describes a reduction from the problems described above to their edge separator, respectively edge-disjoint path packing variants.

We now show our positive result.

Theorem 4.1.8 *Let G be a c-edge-colored graph which is PC acyclic of type 4, and let x,y be arbitrary distinct vertices of G. The minimum PC separator problem and the PC path packing problem for G are both in P, even if no acyclic ordering for G is given. Furthermore, let s be the minimum size of a subset of $V(G)\setminus\{x,y\}$ whose removal eliminates all PC paths between x and y and let t be the maximum number of internally vertex-disjoint PC paths between x and y in G. Then $s=t$.*

Proof The solution to both problems, and the proof of the Menger's theorem analogue, will follow the same basic pattern. Observe that we may delete from G every monochromatic vertex distinct from x and y, since such a vertex cannot be an internal vertex of any PC path. This leaves a graph, say H, where every vertex except x and y is incident to edges of two colors. We show that H is PC acyclic of type 5, and that the first and last vertices of any corresponding vertex ordering are x and y (or y and x), respectively.

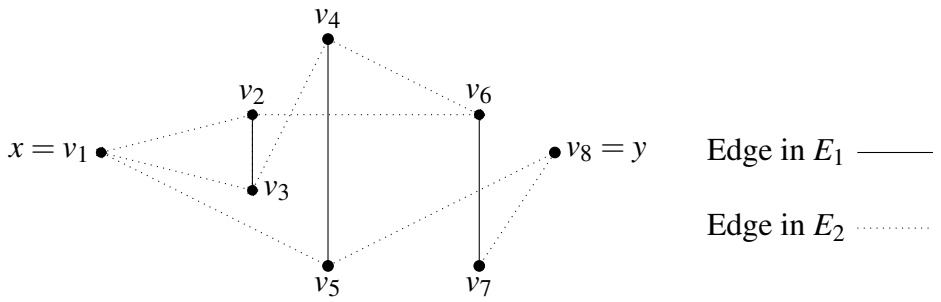

Fig. 4.5 Menger's theorem fails on G

For this, we first note that x and y are both monochromatic in H. Indeed, by assumption G is PC acyclic of type 4, which shows that there exists an induced type 4 ordering for H. The first and last vertices of this ordering must be monochromatic in H; hence these vertices are equal to x and y. Furthermore, it is easy to see that for a graph with only two monochromatic vertices, the notions of being PC acyclic of type 4 and 5 coincide. Thus H is PC acyclic of type 5.

We now proceed as follows. Compute a PC acyclic ordering for H of type 5, and if necessary reverse it so that it begins with the vertex x and ends with y. Note that every PC path from x to y uses the edges of H in the forward direction of the ordering only. Hence we may transform H into a digraph D by orienting every edge of H from its lower-index vertex to its higher-index vertex in the ordering, and solve the corresponding problem on D using well-known polynomial-time algorithms [9]. Menger's theorem also follows from this same reduction. ∎

Finally, we note that Menger's theorem fails to hold if G is only PC acyclic of type 3, even for $c = 2$. Consider the following 2-edge-colored graph G, see Figure 4.5. Let $V(D) = \{v_1, \ldots, v_8\}$, $E_1 = \{v_2v_3, v_4v_5, v_6v_7\}$, $E_2 = \{v_1v_2, v_1v_3, v_1v_5, v_2v_6, v_3v_4, v_4v_6, v_5v_8, v_7v_8\}$. We color edges in E_1 blue and edges in E_2 red. It is easy to check that the ordering $v_8v_7v_6v_1v_2v_3v_4v_5$ of $V(G)$ is PC acyclic of type 3.

Let $x = v_1, y = v_8$, note that any PC path between x and y uses at least two blue edges, thus there is at most one internally vertex-disjoint PC path between x and y. However, after deleting any vertex apart from $\{x, y\}$ the remaining graph will still have a PC path between x and y: after deleting v_2 or v_3, we have PC path $xv_5v_4v_6v_7y$; after deleting v_4 or v_5, we have PC path $xv_3v_2v_6v_7y$; after deleting v_6 or v_7, we have PC path $xv_2v_3v_4v_5y$. Thus $s > t$.

4.2 Odd PC Cycle Detection

One of the central topics in graph theory is the existence of certain kinds of cycles in graphs. In digraphs, it is not hard to decide the existence of any dicycle by simply checking whether a given digraph is acyclic [9]. The problem of existence of PC cycles in edge-colored graphs is less trivial. To solve the problem, we may use Yeo's theorem [109]: if an edge-colored graph G has no PC cycle then G contains a vertex z such that no connected component of $G-z$ is joined to z with edges of more than one color. Thus, we can recursively find such vertices z and delete them from G; if we end up with a trivial graph (containing just one vertex) then G has no PC cycles; otherwise G has a PC cycle. Clearly, the recursive algorithm runs in polynomial time. There is another approach to decide whether there is a PC cycle in a given edge colored graph, which can even find a PC cycle in polynomial time.

Theorem 4.2.1 ([7], Theorem 3.8) *One can construct a maximum cycle subgraph in a c-edge-coloured multigraph G on n vertices in time $O(n^3)$.*

One of the next natural questions is to decide whether a digraph has an odd (even, respectively) dicycle, i.e. a dicycle of odd (even) length, respectively. For odd dicycles we can employ the following well-known result (see, e.g., [9, 52]): A strongly connected digraph is bipartite if and only if it has no odd dicycle. (Note that the result does not hold for non-strongly connected digraphs.) Thus, to decide whether a digraph D has an odd cycle, we can find strongly connected components of D and check whether all components are bipartite. This leads to a simple polynomial-time algorithm. The question of whether we can decide in polynomial time whether a digraph has an even dicycle, is much harder and was an open problem for quite some time till it was solved, in affirmative, independently by McCuaig, and Robertson, Seymour and Thomas (see [83]) who found highly non-trivial proofs.

ODD PC CYCLE DETECTION
Instance: A connected edge colored graph $G=(V,E)$.
Output: Decide if there is an odd PC cycle in G.

In this section we consider the ODD PC CYCLE DETECTION problem. We show that while a natural extension of the odd dicycle solution does not work, an algebraic approach using Tutte matrices and the Schwartz-Zippel lemma allows us to prove that there is a randomized polynomial-time algorithm for solving the problem. The existence of a deterministic polynomial-time algorithm for the odd PC cycle problem remains an open question, as does the existence of a polynomial-time algorithm for the even PC cycle problem.

In this section, we allow our graphs to have multiple edges (but no loops) and call them, for clarity, *multigraphs*. In edge-colored multigraphs, we allow parallel edges of different

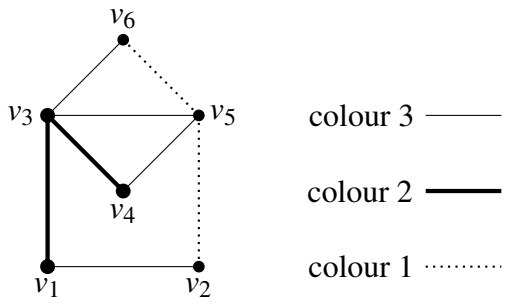

Fig. 4.6 Non-bipartite cyclic connected graph with no odd PC cycle

colors (there is no need to consider parallel edges of the same color). For an edge xy and a vertex v, we use $\chi(xy)$ and $\chi(v)$ to denote the color of xy and the set of colors of edges incident to v, respectively. For any other terminology and notation not provided here, we refer the readers to [9].

4.2.1 Graph-Theoretical Approaches

Recall that to solve the odd dicycle problem, in the previous section, we used the following result: A strongly connected digraph is bipartite if and only if it has no odd dicycle. It is not straightforward to generalize strong connectivity to edge-colored multigraphs. We are aware of two types of connectivity in edge colored graphs. The notion color-connectivity[1], introduced by Saad [84] under another name, does not appear to be useful to us as, in general, it does not partition vertices into components. One could wonder whether every non-bipartite cyclic connected edge-colored graph has an odd PC cycle. Unfortunately, it is not true, see a graph H in Fig. 4.6. It is not hard to check that H is not bipartite and cyclic connected. It has even PC cycles, such as $v_1v_2v_5v_3v_1$, but no odd PC cycles.

Another natural idea is to find some odd PC closed walk first, and hope to find an odd PC cycle in it. Unfortunately, we cannot generate all possible PC closed walks in polynomial time, and moreover a PC closed walk does not necessarily contain an odd PC cycle, see the graph in Figure 4.7. It contains an odd PC walk, but not an odd PC cycle.

4.2.2 Algebraic Approach

In an edge-colored multigraph G, a vertex v is *monochromatic* if $|\chi(v)| = 1$. Let G' be the multigraph obtained from G by recursively deleting monochromatic vertices such that G' has

[1] An edge colored graph G is *color connected* if any two vertices x and y in G are joined by two PEC paths $xx'...y'y$ and $xu...vy$ such that $c(xx') \neq c(xu)$ and $c(y'y) \neq c(vy)$

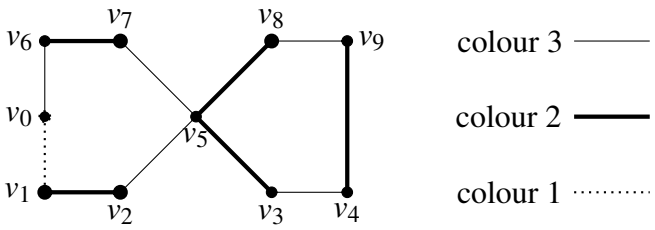

Fig. 4.7 An odd PC closed walk

no monochromatic vertex. Following [95], let G_x, $x \in V(G')$ denote a graph with vertex set

$$\{x_i, x_i' : i \in \chi(x)\} \cup \{x_a'', x_b''\} \text{ and edge set}$$

$$\{x_a''x_b'', x_i'x_a'', x_i'x_b'', x_ix_i' : i \in \chi(x)\}.$$

Let G^* denote a graph with vertex set $\bigcup_{x \in V(G')} V(G_x)$ and edge set $E_1 \cup E_2$, where $E_1 = \bigcup_{x \in V(G')} E(G_x)$ and $E_2 = \{y_q z_q : yz \in E(G'), \chi(yz) = q\}$. Let $c = \max\{\chi(x) : x \in V(G)\}$. Note that

$$|V(G^*)| = O(c|V(G)|) \tag{4.1}$$

A subgraph of an edge-colored multigraph is called a *PC cycle subgraph* if it consists of vertex-disjoint PC cycles. We will use the following result of [47].

Theorem 4.2.2 *Let G be a connected edge-colored multigraph such that G' is non-empty and G^* constructed as above. Then G has a PC cycle subgraph with r edges if and only if G^* has a perfect matching with exactly r edges in E_2.*

Using Theorem 4.2.2, the problem of deciding if there exists an odd PC cycle in G reduces to that of deciding if there is a perfect matching with an odd number of edges from E_2 in the graph G^* (indeed, G^* has an odd PC cycle subgraph if and only if it has an odd PC cycle).

We use the properties of Tutte matrices to solve the reduced problem. For a graph $G = (V, E)$ with $V = \{v_1, v_2, ..., v_n\}$, the *Tutte matrix* A_G is the $n \times n$ multivariate polynomial matrix with entries

$$A_G(i,j) = \begin{cases} x_{ij} & \text{if } v_iv_j \in E \text{ and } i < j \\ -x_{ji} & \text{if } v_iv_j \in E \text{ and } i > j \\ 0 & \text{otherwise,} \end{cases} \tag{4.2}$$

where x_{ij} are distinct variables. Tutte [100] proved that a graph G has a perfect matching if and only if $\det A_G$ is not identically 0.

We say that a matrix A is *skew symmetric* if $A+A^T=0$. Note that the Tutte matrix is skew symmetric. In our argument, we will use the notion of Pfaffian of a skew symmetric matrix. Let $A=[a_{ij}]$ be a $2n\times 2n$ skew symmetric matrix. The *Pfaffian* of A is defined as follows.

$$\mathrm{pf}A=\sum_{\sigma} sgn(\sigma)\prod_{i=1}^{n} a_{\sigma(2i-1),\sigma(2i)}, \tag{4.3}$$

where $sgn(\sigma)$ is the signature of σ and the summation is over all permutations σ such that $\sigma(2i-1)<\sigma(2i)$ for each $1\leq i\leq n$, and $\sigma(2i)<\sigma(2i+2)$ for each $1\leq i<n$ (i.e., each partition of the set $\{1,\ldots,2n\}$ into pairs is included in the sum exactly once). Note in particular that for a graph G, this formula for $\mathrm{pf}A_G$ enumerates every perfect matching of G exactly once.

Observe that, if we regard a_{ij} as indeterminate, $\mathrm{pf}A$ is a multi-linear polynomial. For an odd skew symmetric matrix A, the Pfaffian is defined to be zero. We will use the following well-known relation between Pfaffian and determinants of skew symmetric matrices (see, e.g., [63]).

Theorem 4.2.3 *If A is a skew symmetric matrix, then* $\det A=(\mathrm{pf}A)^2$.

Given a graph $G=(V,E)$ and a subset of edges $E_0\subseteq E(G)$, we now define another skew symmetric matrix A_{G,E_0} whose entries are

$$A_{G,E_0}(i,j)=\begin{cases} -A_G(i,j) & \text{if } v_iv_j\in E_0\\ A_G(i,j) & \text{if } v_iv_j\notin E_0\end{cases} \tag{4.4}$$

It is easy to see that A_{G,E_0} is also a skew symmetric matrix, thus by Theorem 4.2.3, $\det A_{G,E_0}=(\mathrm{pf}A_{G,E_0})^2$. Note that A_G and A_{G,E_0} only differ at entries corresponding to edges in E_0. We call a perfect matching M in a graph G *E_0-odd* (*E_0-even*, respectively) if $|M\cap E_0|$ is odd (even, respectively). Here is our key result.

Lemma 4.2.1 *Given a graph G with even number of vertices and an edge subset $E_0\subseteq E(G)$, let A_G and A_{G,E_0} be defined as in (4.2) and (4.4). Then* $\det A_{G,E_0}=\det A_G$ *if and only if all the perfect matchings of G are of same E_0-parity.*

Proof As both A_G and A_{G,E_0} are skew symmetric, by Theorem 4.2.3,

$$\det A_G=(\mathrm{pf}A_G)^2 \text{ and } \det A_{G,E_0}=(\mathrm{pf}A_{G,E_0})^2.$$

Thus $\det A_{G,E_0}=\det A_G$ if and only if $\mathrm{pf}A_{G,E_0}=\mathrm{pf}A_G$ or $\mathrm{pf}A_{G,E_0}=-\mathrm{pf}A_G$. By (4.3), $\mathrm{pf}A_G$ and $\mathrm{pf}A_{G,E_0}$ both enumerate perfect matchings of G, and by the definitions of A_G and A_{G,E_0},

for each such matching M its contributions to $\mathrm{pf}A_G$ and $\mathrm{pf}A_{G,E_0}$ differ (by a sign term) if and only if M is E_0-odd. Hence $\mathrm{pf}A_{G,E_0} = \mathrm{pf}A_G$ ($\mathrm{pf}A_{G,E_0} = -\mathrm{pf}A_G$, respectively) if and only if each perfect matching in G is E_0-even (E_0-odd, respectively). ∎

For $G = G^*$ and $E_0 = E_2 \subseteq E(G^*)$, by Lemma 4.2.1, if $\det A^*{}_{G,E_2} \neq \det A^*_G$, then G^* has a E_2-odd perfect matching and a E_2-even perfect matching. If $\det A^*{}_{G,E_2} = \det A^*_G$, then every perfect matching of G^* is either E_0-even or E_0-odd. In such a case, we can find a perfect matching M of the graph G^* in polynomial time, and decide the parity of $M \cap E_2$. So we have an algorithm for deciding if G^* has a perfect matching with even or odd number of edges in E_2. Unfortunately, we do not know whether this algorithm is polynomial or not as there is no polynomial algorithm to decide whether a multivariate polynomial is identically zero. Fortunately, we can have a polynomial randomized algorithm due to the following well-known lemma, called the Schwartz-Zippel lemma.[2]

Lemma 4.2.2 *Let $P(x_1, x_2, \ldots, x_n)$ be a multivariate polynomial of total degree at most d over a field F, and assume that P is not identically zero. Pick $r_1, r_2, \ldots, r_n$ uniformly at random from a finite set S of values where $S \subset F$. Then the probability $\mathbb{P}(P(r_1, r_2, \ldots, r_n) = 0) \leq \frac{d}{|S|}$.*

Now it is not hard to prove the following:

Theorem 4.2.4 *Let G be an edge-colored multigraph with n vertices and let $c = \max\{\chi(v) : v \in V(G)\}$. There is a randomized algorithm running in time $O((cn)^\omega)$, where $\omega < 2.3729$, that decides if there is an odd PC cycle in G, with false negative probability less than 1/4.*

Proof As $f(\mathbf{x}) = \det A^*_{G,E_2} - \det A^*_G$ is a multivariate polynomial of degree at most n, we may choose a set S of real values, such that $|S| > 4n$, and use Lemma 4.2.2 to decide if $\det A^*{}_{G,E_2} \neq \det A^*_G$ in polynomial time, with false negative less than 1/4. To see that the running time is $O((cn)^\omega)$, recall (4.1) and observe that computing the determinants of A^*_{G,E_2} and A^*_G will take time $O((cn)^\omega)$, where $\omega < 2.3729$, by the algorithm in [105]. Finally, for the case that $f(\mathbf{x}) \equiv 0$, we can use the algorithm of Mucha and Sankowski [69] to find a perfect matching in time $O((cn)^\omega)$, and then decide its E_0-parity. ∎

[2] It was independently discovered by several authors: Schwartz [86], Zippel [112], DeMillo and Lipton [27].

Chapter 5

Conclusion and Open Problems

In this thesis, we have provided several new results in the area of Parameterized Tractability and in one of its important subfield, kernelization. We conclude the thesis with a list of interesting problems that arise naturally from our results, or to which the techniques developed here might be amenable.

1. in Section 2.1, we obtain an FPT algorithm for k-DCPP. Our algorithms for solving both k-DCPP and k-ADCP have very large running time bounds, mainly because the size bound $f(h^{-1}(k))$ of the minimum feedback arc set is very large. Function $f(k)$ obtained in [81] is a multiply iterated exponential, where the number of iterations is also a multiply iterated exponential and, as a result, $h^{-1}(k)$ grows very quickly. So if we can obtain a significantly smaller upper bound for $f(k)$ on Euler digraphs would significantly reduce $h^{-1}(k)$ as well and is of certain interest itself. In particular, we would like to know whether it is true that $f(k) = O(k^{O(1)})$ for Euler digraphs? Note that for planar digraphs, $f(k) = k$ [9, Corollary 15.3.10] and Seymour [87] proved the same result for a wide family of Euler digraphs. Utilizing this property, we can design a much faster FPT algorithm for k-DCPP on planar graph, see 2.1.4. A natural question would be can we provide a polynomial size kernel for k-DCPP, at least on planar graphs. If one succeeds in this, it would be interesting to further check whether the k-DCPP and k-ADCP admits a polynomial size kernel on general directed graphs.

2. in Section 2.2, we solve an open problem asked in [102] by showing that MCPP parameterized with the number of arcs is fixed-parameter tractable. To prove this result we reduce MCPP to BCPP, a generalization of UCPP, and apply a very useful lemma of Marx *et al.* [66], which bounds the treewidth of the torso graph with respect to small separators. Note that our application of the lemma is significantly different

from those in [66] and we believe that our approach will be of interest in designing fixed-parameter algorithms for other problems.

van Bevern *et al.* [102] mentioned two other parameterizations of MCPP. One of them is by the treewidth of the given graph G. This question was answered by Gutin *et al.* in [46] by proving this parameterization (even with pathwidth) of MCPP is W[1]-hard. They also showed that MCPP parameterized with treedepth is FPT. They further suggested other parameters to study, like *distance to linear forest*. A vertex v of G is called even, if the number of arcs and edges incident to v is even. Edmonds and Johnson [30] proved that if all vertices of G are even, then MCPP is polynomial time solvable on G. So, the number of odd (not even) vertices is a natural parameter. It is still open whether MCPP parameterized with the number of odd vertices is FPT [102].

3. in Section 2.3, we consider the Chinese Postman Problem on edge-colored graphs (CPP-ECG). This problem generalizes the Chinese Postman Problem on undirected and directed graphs and the properly colored Euler trail problem on edge-colored graphs, all of which can be solved in polynomial time. We proved that CPP-ECG is still polynomial time solvable, enhancing our understanding of the Chinese Postman Problem.

   It is well-known that the number of Euler trails on digraphs can be calculated in polynomial time using the so-called BEST theorem [99, 101], named after de Bruijn, van Aardenne-Ehrenfest, Smith and Tutte. However, the problem turns out to be much harder on undirected graphs as it is provd to be #P-complete [20]. Our simple transformation from directed walks to PC walks described in the beginning of Section 2.3, shows that the problem of counting PC Euler trails on 2-edge-colored graphs generalizes that of counting the number of Euler trails on digraphs. Assigning each edge of an undirected graph a distinct color, shows that the problem of counting PC Euler trails on k-edge-colored graphs is #P-complete when k is unbounded. So it would be interesting to determine the complexity of the problem of counting PC Euler trails on a k-edge-colored graphs when k is bounded, in particular when $k = 2$.

4. in Section 3.1, we study the c-LOAD COLORING problem. We believe that our bound on the number of vertices in a kernel is optimal, but the bound on the number of edges may not be optimal even for $c = 2$. We conjecture that the optimal bound is $c^2k + O(1)$. Here is an example showing its tightness. Consider the complete bipartite graph $K_{c,ck-1}$ and add all possible edges between vertices of the partite set of size c. The resulting digraph is a reduced No-instance with $c^2k + c(c-3)/2$ edges.

Our linear-vertex kernel result implies an $O^*(c^{2ck})$-time algorithm for c-LOAD COLORING, which simply tests all the c-colorings of the kernel. It is however possible that the problem admits much better FPT algorithms, since the complement of c-LOAD COLORING, NO c-LOAD COLORING, has small, but not constant, forbidden minors and is minor-bidimensional (see [25, 26] for more information on forbidden minors and bidimensionality).

Let $pw(G)$ and $tw(G)$ denote the pathwidth and the treewidth of G. Since the path $P_{c(k+1)}$ is one of the forbidden minors for NO c-LOAD COLORING, it is easy to decide whether $G \in (c,k)$-LC or G has a path-decomposition of size bounded by $c(k+1)$. Indeed, if $G \notin (c,k)$-LC, any DFS on the connected components of G gives a Tremaux tree with depth bounded by $c(k+1)$ that we may transform into path decomposition of size bounded by $c(k+1)$ in polynomial time. Since the $O^*(2^{tw(G)})$-time algorithm for 2-LOAD COLORING from [41] can be generalized to an $O^*(c^{tw(G)})$-time algorithm for c-LOAD COLORING, there exists a $O^*(c^{ck})$-time algorithm for this problem.

For $c = 2$, the running time $O^*(4^k)$ (first obtained in [41]) can be improved using the result by Kneis et al. [55] that a graph with m edges and n vertices has treewidth at most $m/5.769 + O(\log n)$. Thus, by Theorem 3.1.3 in polynomial time we can reduce a graph G to a graph G' with $tw(G') \leq 1.0401k + O(\sqrt{k})$. Therefore, the $O^*(2^{tw(G)})$ algorithm for 2-LOAD COLORING has running time $O^*(2.0564^k)$.

If we require that the input G is H-minor-free for some fixed graph H, then $tw(G) = O(\sqrt{n})$ by [25, 26], and our linear-vertex kernel leads to an $O^*(c^{O(\sqrt{ck})})$-time algorithm. Unfortunately, there is no constant forbidden minor for NO c-LOAD COLORING as membership in (c,k)-LC requires at least ck edges.

Nevertheless, by Theorem 4.12 of [25], and since branchwidth is linked to the treewidth up to a constant factor, any graph G contains an $(\Omega(\frac{tw(G)}{gen(G)}) \times \Omega(\frac{tw(G)}{gen(G)}))$-grid as a minor, where $gen(G)$ is the genus of G. Since the $(r \times r)$-grid is a forbidden minor for NO c-LOAD COLORING when $r \geq \lceil \sqrt{(c+1)k} \rceil$, we have $tw(G) = O(\sqrt{ck}\, gen(G))$. Thus, we obtain an $O^*(c^{O(\sqrt{ck}\, gen(G))})$-time algorithm to solve c-LOAD COLORING, which is subexponential for graphs of bounded genus. Note also that the complete graph $K_{c\lceil\sqrt{2k}+1\rceil}$ is also one of the forbidden minors. Thus, the Hadwiger number $h(G)$ of G is bounded by $c\lceil\sqrt{2k}+1\rceil$. For any family with treewidth bounded by $o(h(G)^2)$, there is a subexponential algorithm. For instance, there is an $O^*(c^{\sqrt{ck}})$-time algorithm for chordal graphs.

Let us discuss extensions of our results to graphs that may have loops and multiple edges. Let us start with isolated vertices and loops. Since the isolated vertices do not

involve any edges, it is safe to delete them. Observe that loops are always colored with the color of their vertex. But a leaf has also to be colored with the color of its neighbor, as otherwise the edge between them is uncolored. Thus, any loop can be replaced by a pendant edge.

It remains to consider the multiple edges. Since multiple edges can be colored with at most one color, it is safe to reduce any multigraph with multiplicity greater than k to its maximal induced subgraph with multiplicity k. It is not hard to show that the overloads from Definition 3.1.1 can be generalized just by requiring that for all $u \in V_1, |E(\{u\}, V_u)| \geq k$. Thus, the reductions are also safe for multigraphs. As the maximal induced (simple) graph of any multigraph has the same number of vertices and the same connectivity, our bound on the number of vertices in a kernel holds for multigraphs, too. The bounds on the number of edges in a kernel has to be slightly changed. Let t be the maximal multiplicity of an edge in the multigraph under consideration. Then the bound of the number of edges in a kernel (for any c) will be $6.25c^2tk$ and thus we will have an approximation algorithm of ratio $12.5ct$.

5. In Section 3.2, we discuss the problem of packing stars into a graph. We obtain a linear kernel for the problem on graphs with no long induced paths. This result was later improved by Mingyu Xiao [107], who obtained a linear kernel for the problem on general graphs.

6. in Section 4.1, we introduce and study five types of PC acyclicity in edge-colored graphs such that graphs of PC acyclicity of type i is a proper superset of graphs of acyclicity of type $i+1$, $i = 1,2,3,4$. Using the five types, we investigate the border between intractability and tractability for the problems of finding the maximum number of internally vertex-disjoint PC paths between two vertices and the minimum number of vertices to meet all PC paths between two vertices. Consider the problem of deleting as few vertices as possible from a 2-edge-colored graph to get a subgraph of PC acyclicity of type 5. We will show that this problem generalizes both the directed feedback vertex set problem in digraphs and the bipartization problem. In the directed feedback vertex set problem, given a digraph D, we want to find a minimum size vertex set S such that $D - S$ has no directed cycle. In the bipartization problem, given an undirected graph G, one is asked to find a minimum size vertex set S such that $G - S$ is bipartite. Both directed feedback vertex problem and bipartization problem are NP-hard but fixed-parameter tractable with respect to the parameter $|S|$ [24, 29]. Thus, the problem of vertex deletion to a PC acyclic 2-edge-colored graph of type 5, is NP-hard, but we

do not know whether it is fixed-parameter tractable with respect to the minimum size of a solution.

For a digraph D, let G be the 2-edge-colored graph obtained by duplicating every vertex $v \in D$, to v' and v'' and adding a red edge between v' and v''. Then for every arc uv in D we add the blue edge $u''v'$ to G. This completes the description of G. Clearly G is bipartite. If S is a minimum vertex set of G such that $G-S$ is PC acyclic of type 5, then by Theorem 4.1.2 $G-S$ has no PC cycle. By Theorem 4.1.2 and the minimality of S we may assume that $S \subseteq V(D)' := \{v' : v \in V(D)\}$ (if $v'' \in S$ we can take v' and not v''). This implies that $D-S$ has no directed cycle (if it did we would have a PC cycle in $G-S$). Conversely, if T is a minimum feedback vertex set in D, then $D-T$ has no directed cycle and it is not difficult to see that $G-T'$ has no PC cycle. As $G-T'$ is bipartite and has no PC cycle, by Corollary 4.1.2 it is of type 5. Thus, our problem generalizes the directed feedback vertex set problem in digraphs.

If G is a graph, then let the edge-colored graph G' be equal to G, where all edges have same color. If $G-S$ is bipartite then $G'-S$ is bipartite and has no PC cycle (as all edges have the same color) and therefore $G'-S$ is PC acyclic of type 5. Conversely, if $G'-S$ is PC acyclic of type 5, then $G'-S$ is bipartite and $G-S$ is bipartite. Thus, our problem generalizes the bipartization problem.

Moreover, our problem is more general, as there are many 2-edge-colored graphs that do not arise from directed graphs using the standard transformation given in the beginning of Chapter 4.

7. In Section 4.2, we provide a randomized polynomial time algorithm for the odd PC cycle problem. A naturally following question is whether this problem can be solved in (deterministic) polynomial time. We believe that the answer to the question is positive, like with some other problems. For example, Yannis Manoussakis [65] asked whether there is a polynomial-time algorithm to find a longest alternating cycle in 2-edge-colored complete graphs. Saad [84] first designed a randomized polynomial-time algorithm for the problem and later Bang-Jensen and Gutin [8] answered the question of Manoussakis in affirmative.

   Another interesting problem to study is whether one can decide the existence of an even PC cycle in polynomial time.

   It was proved in [44] that if an edge-colored graph G has no PC closed walk then G has a monochromatic vertex. This can be viewed as a characterization of edge-colored graphs with no PC closed walk and it implies that deciding whether G has a PC closed walk is polynomial-time solvable. We are unware of a precise characterization of

edge-colored graphs with no odd PC walk, since they differ from graphs with no odd PC cycle, following the discussion in Section 4.2.2. We leave open the problem of deciding whether an edge-colored graph has an odd PC walk.

References

[1] Abouelaoualim, A., Das, K. C., de la Vega, W. F., Karpinski, M., Manoussakis, Y., Martinhon, C., and Saad, R. (2010). Cycles and paths in edge-colored graphs with given degrees. *Journal of Graph Theory*, 64(1):63–86.

[2] Abouelaoualim, A., Das, K. C., Faria, L., Manoussakis, Y., Martinhon, C., and Saad, R. (2008). Paths and trails in edge-colored graphs. *Theoretical computer science*, 409(3):497–510.

[3] Ahuja, N., Baltz, A., Doerr, B., Přívětivỳ, A., and Srivastav, A. (2005). On the minimum load coloring problem. In *Approximation and Online Algorithms*, pages 15–26. Springer.

[4] Ahuja, S. K. (2010). Algorithms for routing and channel assignment in wireless infrastructure networks.

[5] Allouche, J.-P. and Shallit, J. (1992). The ring of k-regular sequences. *Theoretical Computer Science*, 98(2):163–197.

[6] Arnborg, S., Corneil, D. G., and Proskurowski, A. (1987). Complexity of finding embeddings in ak-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284.

[7] Bang-Jensen, J. and Gutin, G. (1997). Alternating cycles and paths in edge-coloured multigraphs: a survey. *Discrete Mathematics*, 165:39–60.

[8] Bang-Jensen, J. and Gutin, G. (1998). Alternating cycles and trails in 2-edge-coloured complete multigraphs. *Discrete Mathematics*, 188(1):61–72.

[9] Bang-Jensen, J. and Gutin, G. Z. (2009). *Digraphs: theory, algorithms and applications*. Springer Science & Business Media, 2 edition.

[10] Bar-Yehuda, R., Halldórsson, M. M., Naor, J., Shachnai, H., and Shapira, I. (2006). Scheduling split intervals. *SIAM Journal on Computing*, 36(1):1–15.

[11] Barbero, F., Gutin, G., Jones, M., and Sheng, B. (2015). Parameterized and Approximation Algorithms for the Load Coloring Problem. In Husfeldt, T. and Kanj, I., editors, *10th International Symposium on Parameterized and Exact Computation (IPEC 2015)*, volume 43 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 43–54. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[12] Barbero, F., Gutin, G., Jones, M., Sheng, B., and Yeo, A. (2016). Linear-vertex kernel for the problem of packing r-stars into a graph without long induced paths. *Information Processing Letters*, 116(6):433–436.

[13] Bejar, R., Krishnamachari, B., Gomes, C., and Selman, B. (2001). Distributed constraint satisfaction in a wireless sensor tracking system. In *Workshop on Distributed Constraint Reasoning, International Joint Conference on Artificial Intelligence*, volume 4.

[14] Beltrami, E. J. and Bodin, L. D. (1974). Networks and vehicle routing for municipal waste collection. *Networks*, 4(1):65–94.

[15] Benkouar, A., Manoussakis, Y., Paschos, V. T., and Saad, R. (1991). On the complexity of some hamiltonian and eulerian problems in edge-colored complete graphs. In *International Symposium on Algorithms*, pages 190–198. Springer.

[16] Bodlaender, H. L. (1996). A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on computing*, 25(6):1305–1317.

[17] Bodlaender, H. L., Fomin, F. V., Lokshtanov, D., Penninkx, E., Saurabh, S., and Thilikos, D. M. (2009). (meta) kernelization. In *Foundations of Computer Science, 2009. FOCS'09. 50th Annual IEEE Symposium on*, pages 629–638. IEEE.

[18] Bollobás, B. (2013). *Modern graph theory*, volume 184. Springer Science & Business Media.

[19] Brandstädt, A., Le, V. B., and Spinrad, J. P. (1999). *Graph classes: a survey*. SIAM.

[20] Brightwell, G. R. and Winkler, P. (2004). Note on counting eulerian circuits. *lanl. arXiv. org*.

[21] Brucker, P. (1981). The chinese postman problem for mixed graphs. ingraphtheoretic concepts in computer science (pp. 354-366).

[22] Chen, J., Liu, Y., Lu, S., O'sullivan, B., and Razgon, I. (2008). A fixed-parameter algorithm for the directed feedback vertex set problem. *Journal of the ACM (JACM)*, 55(5):21.

[23] Christofides, N. (1973). The optimum traversal of a graph. *Omega*, 1(6):719–732.

[24] Cygan, M., Fomin, F. V., Kowalik, Ł., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., and Saurabh, S. (2015). *Parameterized Algorithms*, volume 4. Springer.

[25] Demaine, E. D., Fomin, F. V., Hajiaghayi, M., and Thilikos, D. M. (2005). Subexponential parameterized algorithms on bounded-genus graphs and h-minor-free graphs. *Journal of the ACM (JACM)*, 52(6):866–893.

[26] Demaine, E. D. and Hajiaghayi, M. (2008). The bidimensionality theory and its algorithmic applications. *The Computer Journal*, 51(3):292–302.

[27] DeMillo, R. A. and Lipton, R. J. (1978). A probabilistic remark on algebraic program testing. *Information Processing Letters*, 7(4):193–195.

[28] Dorn, F., Moser, H., Niedermeier, R., and Weller, M. (2013). Efficient algorithms for eulerian extension and rural postman. *SIAM Journal on Discrete Mathematics*, 27(1):75–94.

[29] Downey, R. G. and Fellows, M. R. (2013). *Fundamentals of parameterized complexity*, volume 4. Springer.

[30] Edmonds, J. and Johnson, E. L. (1973). Matching, euler tours and the chinese postman. *Mathematical programming*, 5(1):88–124.

[31] Eiselt, H. A., Gendreau, M., and Laporte, G. (1995). Arc routing problems, part i: The chinese postman problem. *Operations Research*, 43(2):231–242.

[32] Fellows, M., Heggernes, P., Rosamond, F., Sloper, C., and Telle, J. A. (2004). Finding k disjoint triangles in an arbitrary graph. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 235–244. Springer.

[33] Fellows, M. R., Guo, J., Moser, H., and Niedermeier, R. (2011). A generalization of nemhauser and trotters local optimization theorem. *Journal of Computer and System Sciences*, 77(6):1141–1158.

[34] Fleischner, H., Sabidussi, G., and Wenger, E. (1992). Transforming eulerian trails. *Discrete mathematics*, 109(1-3):103–116.

[35] Flum, J. and Grohe, M. (2006). Parameterized complexity theory, volume xiv of texts in theoretical computer science. an eatcs series.

[36] Fujita, S. and Magnant, C. (2011). Properly colored paths and cycles. *Discrete Applied Mathematics*, 159(14):1391–1397.

[37] Garey, M. R. and Johnson, D. S. (1990). *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA.

[38] Grohe, M. and Grüber, M. (2007). Parameterized approximability of the disjoint cycle problem. In *International Colloquium on Automata, Languages, and Programming*, pages 363–374. Springer.

[39] Grossman, J. W. and Häggkvist, R. (1983). Alternating cycles in edge-partitioned graphs. *Journal of Combinatorial Theory, Series B*, 34(1):77–81.

[40] GUAN, M. (1962). Graphic programming using odd or even points. *Chinese Mathematics*, 1:237–277.

[41] Gutin, G. and Jones, M. (2014). Parameterized algorithms for load coloring problem. *Information Processing Letters*, 114(8):446–449.

[42] Gutin, G., Jones, M., and Sheng, B. (2014a). Parameterized complexity of the k-arc chinese postman problem. In *European Symposium on Algorithms*, pages 530–541. Springer.

[43] Gutin, G., Jones, M., Sheng, B., and Wahlström, M. (2014b). Parameterized directed k-chinese postman problem and k arc-disjoint cycles problem on euler digraphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 250–262. Springer.

[44] Gutin, G., Jones, M., Sheng, B., Wahlström, M., and Yeo, A. (2017a). Acyclicity in edge-colored graphs. *Discrete Mathematics*, 340(2):1–8.

[45] Gutin, G., Jones, M., Sheng, B., Wahlström, M., and Yeo, A. (2017b). Chinese postman problem on edge-colored multigraphs. *Discrete Applied Mathematics*, 217:196–202.

[46] Gutin, G., Jones, M., and Wahlström, M. (2015). Structural parameterizations of the mixed chinese postman problem. In *Algorithms-ESA 2015*, pages 668–679. Springer.

[47] Gutin, G. and Kim, E. J. (2009). Properly coloured cycles and paths: results and open problems. In *Graph theory, computational intelligence and thought*, pages 200–208. Springer.

[48] Gutin, G., Muciaccia, G., and Yeo, A. (2013). Parameterized complexity of k-chinese postman problem. *Theoretical Computer Science*, 513:124–128.

[49] Gutin, G., Rafiey, A., Szeider, S., and Yeo, A. (2007). The linear arrangement problem parameterized above guaranteed value. *Theory of Computing Systems*, 41(3):521–538.

[50] Gutin, G., Sheng, B., and Wahlström, M. (2016). Odd properly colored cycles in edge-colored graphs. *Discrete Mathematics*.

[51] Gutin, G., Szeider, S., and Yeo, A. (2008). Fixed-parameter complexity of minimum profile problems. *Algorithmica*, 52(2):133–152.

[52] Harary, F., Norman, R., and Cartwright, D. (1965). Structural models: An introduction to the theory of directed graphs. john willey & sons. *Inc., New York. MATH*.

[53] Kirkpatrick, D. G. and Hell, P. (1978). On the completeness of a generalized matching problem. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 240–245. ACM.

[54] Kloks, T. (1994). *Treewidth: computations and approximations*, volume 842. Springer Science & Business Media.

[55] Kneis, J., Mölle, D., Richter, S., and Rossmanith, P. (2009). A bound on the pathwidth of sparse graphs with applications to exact algorithms. *SIAM Journal on Discrete Mathematics*, 23(1):407–427.

[56] Korte, B. and Vygen, J. (2005). Combinatorial optimization. theory and algorithms. 2000. *Cited on*, page 84.

[57] Kotzig, A. (1968). Moves without forbidden transitions in a graph. *Matematickỳ časopis*, 18(1):76–80.

[58] Lo, A. (2012). Properly coloured hamiltonian cycles in edge-coloured complete graphs. *Combinatorica*, pages 1–22.

[59] Lo, A. (2014a). A dirac type condition for properly coloured paths and cycles. *Journal of Graph Theory*, 76(1):60–87.

[60] Lo, A. (2014b). An edge-colored version of dirac's theorem. *SIAM Journal on Discrete Mathematics*, 28(1):18–36.

[61] Loebl, M. and Poljak, S. (1990). Subgraph packing—a survey. In *Topics in Combinatorics and Graph Theory*, pages 491–503. Springer.

[62] Lokshtanov, D., Misra, N., and Saurabh, S. (2012). Kernelization-preprocessing with a guarantee. In *The Multivariate Algorithmic Revolution and Beyond*, pages 129–161. Springer.

[63] Lovász, L. and Plummer, M. D. (2009). *Matching theory*, volume 367. American Mathematical Soc.

[64] Lucchesi, C. and Younger, D. (1978). A minimax theorem for directed graphs. *J. London Math. Soc.(2)*, 17(3):369–374.

[65] Manoussakis, Y. (1990). On the complexity of finding alternating paths in edge coloured complete graphs. Technical report, University of Paris.

[66] Marx, D., O'sullivan, B., and Razgon, I. (2013). Finding small separators in linear time via treewidth reduction. *ACM Transactions on Algorithms (TALG)*, 9(4):30.

[67] Menger, K. (1927). Zur allgemeinen kurventheorie. *Fundamenta Mathematicae*, 10(1):96–115.

[68] Minieka, E. (1979). The chinese postman problem for mixed networks. *Management Science*, 25(7):643–648.

[69] Mucha, M. and Sankowski, P. (2004). Maximum matchings via gaussian elimination. In *Foundations of Computer Science, 2004. Proceedings. 45th Annual IEEE Symposium on*, pages 248–255. IEEE.

[70] Niedermeier, R. (2006). Invitation to fixed-parameter algorithms.

[71] Opatrny, J. (1979). Total ordering problem. *SIAM Journal on Computing*, 8(1):111–114.

[72] Papadimitriou, C. H. (1976). On the complexity of edge traversing. *Journal of the ACM (JACM)*, 23(3):544–554.

[73] Pearn, W. L. (1994). Solvable cases of the k-person chinese postman problem. *Operations Research Letters*, 16(4):241–244.

[74] Peng, Y. (1989). Approximation algorithms for some postman problems over mixed graphs. *Chinese J. Oper. Res*, 8(1):76–80.

[75] Pevzner, P. (2000a). Computational molecular biology: an algorithmic approach. *Cambridge, Mass.: MIT Press*, 18(3):1.

[76] Pevzner, P. (2000b). *Computational molecular biology: an algorithmic approach*. MIT press.

[77] Pevzner, P. A. (1995). Dna physical mapping and alternating eulerian cycles in colored graphs. *Algorithmica*, 13(1-2):77–105.

[78] Plaisted, D. A. and Zaks, S. (1980). An np-complete matching problem. *Discrete Applied Mathematics*, 2(1):65–72.

[79] Prieto, E. (2005). The method of extremal structure on the k-maximum cut problem. In *Proceedings of the 2005 Australasian symposium on Theory of computing-Volume 41*, pages 119–126. Australian Computer Society, Inc.

[80] Prieto, E. and Sloper, C. (2006). Looking at the stars. *Theoretical Computer Science*, 351(3):437–445.

[81] Reed, B., Robertson, N., Seymour, P., and Thomas, R. (1996). Packing directed circuits. *Combinatorica*, 16(4):535–554.

[82] Robertson, N. and Seymour, P. D. (1986). Graph minors. ii. algorithmic aspects of tree-width. *Journal of algorithms*, 7(3):309–322.

[83] Robertson, N., Seymour, P. D., and Thomas, R. (1999). Permanents, pfaffian orientations, and even directed circuits. *Annals of Mathematics*, 150(3):929–975.

[84] Saad, R. (1996). Finding a longest alternating cycle in a 2-edge-coloured complete graph is in rp. *Combinatorics, Probability and Computing*, 5(03):297–306.

[85] Sankararaman, S., Efrat, A., Ramasubramanian, S., and Agarwal, P. K. (2014). On channel-discontinuity-constraint routing in wireless networks. *Ad hoc networks*, 13:153–169.

[86] Schwartz, J. T. (1980). Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM (JACM)*, 27(4):701–717.

[87] Seymour, P. D. (1996). Packing circuits in eulerian digraphs. *Combinatorica*, 16(2):223–231.

[88] Shallit, J. (2002). Title of citation. http://oeis.org/A073121.

[89] Slivkins, A. (2010). Parameterized tractability of edge-disjoint paths on directed acyclic graphs. *SIAM Journal on Discrete Mathematics*, 24(1):146–157.

[90] Sorge, M. (2013). Some algorithmic challenges in arc routing. In *Talk at NII Shonan Seminar*, volume 18.

[91] Sorge, M., Van Bevern, R., Niedermeier, R., and Weller, M. (2011). From few components to an eulerian graph by adding arcs. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 307–318. Springer.

[92] Sorge, M., Van Bevern, R., Niedermeier, R., and Weller, M. (2012). A new view on rural postman based on eulerian extension and matching. *Journal of Discrete Algorithms*, 16:12–33.

[93] Szachniuk, M., De Cola, M. C., Felici, G., and Blazewicz, J. (2014). The orderly colored longest path problem-a survey of applications and new algorithms. *RAIRO-Operations Research*, 48(1):25–51.

[94] Szachniuk, M., Popenda, M., Adamiak, R. W., and Blazewicz, J. (2009). An assignment walk through 3d nmr spectrum. In *Computational Intelligence in Bioinformatics and Computational Biology, 2009. CIBCB'09. IEEE Symposium on*, pages 215–219. IEEE.

[95] Szeider, S. (2003). Finding paths in graphs avoiding forbidden transitions. *Discrete Applied Mathematics*, 126(2):261–273.

[96] Tardos, É. (1985). A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5(3):247–255.

[97] Thomassen, C. (1997). On the complexity of finding a minimum cycle cover of a graph. *SIAM Journal on Computing*, 26(3):675–677.

[98] Tseng, I.-L., Chen, H.-W., and Lee, C.-I. (2010). Obstacle-aware longest-path routing with parallel milp solvers. In *World Congress on Engineering and Computer Science (WCECS)*, volume 2. Citeseer.

[99] Tutte, W. and Smith, C. (1941). On unicursal paths in a network of degree 4. *The American Mathematical Monthly*, 48(4):233–237.

[100] Tutte, W. T. (1947). The factorization of linear graphs. *Journal of the London Mathematical Society*, 1(2):107–111.

[101] van Aardenne-Ehrenfest, T. and de Bruijn, N. G. (2009). Circuits and trees in oriented linear graphs. In *Classic papers in combinatorics*, pages 149–163. Springer.

[102] van Bevern, R., Niedermeier, R., Sorge, M., and Weller, M. (2014). Complexity of arc routing problems. *Arc Routing: Problems, Methods, and Applications*, 20:19.

[103] Vygen, J. (1995). Np-completeness of some edge-disjoint paths problems. *Discrete Applied Mathematics*, 61(1):83–90.

[104] Wang, J., Ning, D., Feng, Q., and Chen, J. (2008). An improved parameterized algorithm for a generalized matching problem. In *International Conference on Theory and Applications of Models of Computation*, pages 212–222. Springer.

[105] Williams, V. V. (2014). Multiplying matrices in o (n2. 373) time. *preprint*.

[106] Win, Z. (1989). On the windy postman problem on eulerian graphs. *Mathematical Programming*, 44(1):97–112.

[107] Xiao, M. (2017). On a generalization of nemhauser and trotter's local optimization theorem. *Journal of Computer and System Sciences*, 84:97–106.

[108] Yaxiong, L. and Yongchang, Z. (1988). A new algorithm for the directed chinese postman problem. *Computers & operations research*, 15(6):577–584.

[109] Yeo, A. (1997). A note on alternating cycles in edge-coloured graphs. *Journal of Combinatorial Theory, Series B*, 69(2):222–225.

[110] Zaragoza Martinez, F. J. (2003). Postman problems on mixed graphs.

[111] Zhang, L. (1992). Polynomial algorithms for the k-chinese postman problem. In *Proceedings of the IFIP 12th World Computer Congress on Algorithms, Software, Architecture-Information Processing'92, Volume 1-Volume I*, pages 430–435. North-Holland Publishing Co.

[112] Zippel, R. (1979). *Probabilistic algorithms for sparse polynomials*. Springer.