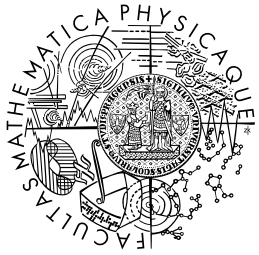

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

DOCTORAL THESIS

Radek Hušek

Structure of Flow-continuous Mappings in Algebraic Context

Computer Science Institute

Supervisor of the doctoral thesis: doc. Mgr. Robert Šámal Ph.D.

Study programme: Computer Science

Study branch: Theory of Computing, Discrete Models and Optimization

Prague 2022

I declare that I carried out this doctoral thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In Prague date ....

Author's signature

First I thank to my supervisor Robert Šámal for his guidance and patience. I also thank to my family – and foremost to my girlfriend Anička – for their support.

Title: Structure of Flow-continuous Mappings in Algebraic Context

Author: Radek Hušek

Institute: Computer Science Institute

Supervisor: doc. Mgr. Robert Šámal Ph.D., Computer Science Institute

Abstract: We explore the structure of the cycle space of the graphs – most notably questions about nowhere-zero flows and cycle double covers. We touch several facets of this field. First we show that there are edge 2-connected graphs which distinguish $\mathbb{Z}_2^2$- and $\mathbb{Z}_4$-connectivity (group connectivity which is a strengthening of nowhere-zero flows).

Then we examine a conjecture of Matt DeVos which asserts existence of group flows given existence of a graph homomorphism between suitable Cayley graphs. We introduce a strengthening of this conjecture called strong homomorphism property (SHP for short) which allows splitting vertices (and hence a reduction to cubic graphs). We conjecture that SHP holds for every graph and the smallest group in which the graph has a nowhere-zero flow and we prove that both SHP and the original conjecture imply existence of cycle double covers with few cycles.

The question we discuss the most is counting objects on graphs – especially counting circuit double covers. We shows an almost exponential lower bound for graphs on surfaces with nice embeddings and we also show that this bound does not apply to Flower snarks. Then we shows quite precise bound for flower snarks and we also improve the lower bound for planar graphs to an exponential one. Along the way we build a framework for counting objects called linear representations which might be of independent interest. We conclude with description of voltage graphs and how to use them to find a new infinite family of snarks.

Keywords: graph, cycle double cover, group flow

Contents

Introduction

The main topic of this thesis is the structure of the cycle space of the graphs – most notably questions about nowhere-zero flows and cycle double covers. We explore structural questions about graphs and they have a strong ties to algebra.

In the first chapter we present the basic notions together with a brief overview of the area. Chapter 2 explores group connectivity which is a strengthening of nowhere-zero flows. It is based on paper Hušek et al. [2019]. Its main result is Theorem 2.2 which states that there are edge 2-connected graphs which distinguish $\mathbb{Z}_2^2$- and $\mathbb{Z}_4$-connectivity. This work was later extended by Han et al. [2020] to edge 3-connected graphs (Theorem 2.13).

Chapter 3 is based on paper Hušek and Šámal [2020] and it examines a conjecture of Matt DeVos which asserts existence of group flows given existence of a graph homomorphism between suitable Cayley graphs (Conjecture 3.4). We introduce a strengthening of this conjecture called strong homomorphism property (SHP for short) which allows splitting vertices (and hence a reduction to cubic graphs). We show that SHP holds for groups of size at most four (Theorem 3.17) and we show a graph for which it does not hold for $\mathbb{Z}_5$ (Figure 3.1). But this graph is not a snark so we conjecture that SHP holds for every graph and the smallest group in which the graph has a nowhere-zero flow (Conjecture 3.12). We conclude the chapter with a proof that both SHP and the original conjecture imply existence of cycle double covers with few cycles (Corollary 3.19).

The next three chapters dive into the topic of counting objects on graphs – in particular counting circuit double covers. A very brief summary of these chapters was presented at EuroComb 2021 as Hušek and Šámal [2021]; the full paper is being prepared. Chapter 4 provides an introduction into the topic and shows an almost exponential lower bound for graphs on surfaces with nice embeddings (Corollary 4.14). Chapter 6 dives deeper and shows quite precise bound for flower snarks (Theorem 6.8) for which the lower bound from the previous chapter does not apply. We also improve the lower bound for planar graphs to an exponential one (Theorem 6.14).

Chapter 5 builds a formal framework for counting objects on graphs using linear algebra which we call linear representations. Aside from the necessary definitions it presents a notion of the best representation (Theorem 5.18). We prove that the best representation is unique up to isomorphism (Observation 5.23). Corollary 5.28 shows a way to test whether a representation is the best one. This framework is used throughout Chapter 6 and also in Chapter 7.

The last chapter (Chapter 7) describes an interesting way to find new graphs. The method is called voltage graphs. We describe the method, our implementation of it and present a new infinite family of snarks obtained by it. Our research presented in this chapter is not finished yet although the preliminary results look very promising. Because of that it was not published yet.

1. Preliminaries and History

We start with a short review of the field of group flows and circuit double covers. This chapter is mostly based on the folklore knowledge and Zhang's books [1997, 2012]. We assume a basic knowledge of the graph theory – if needed see, e.g., Diestel [2017] or any other textbook on this topic. Presented observation are usually a folklore knowledge with simple proofs and as such they are not cited.

1.1 Basic Definitions

Let us start with some basic definitions as some of them use different names then it is common in the other areas of the graph theory and some need slight modifications to work reasonably in the context of counting objects. The probably most notable difference is using the word "cycle" for graphs with degrees of all vertices even and "circuit" for a graph isomorphic to some C_k which has roots in matroid theory.[1]

Definition 1.1 (Circuit and Cycle)**.** *Let G be a graph. A subgraph H of G is a* circuit *if it is isomorphic to C_k for some k. A* cycle *is union of edge-disjoint circuits.*

Equivalently a circuit is a 2-regular connected graphs and a cycle is a graph with degrees of all vertices even. We also need some measure of connectedness of a graph. The usual measure is vertex or edge connectivity. It is not good enough for us because a cubic graph cannot have connectivity more than three and we are mainly interested in cubic graphs. We can require that the only cuts of size three are the ones around vertices. This leads to a refinement of the edge connectivity called cyclic edge connectivity.

Definition 1.2 (Cyclic Connectivity)**.** *A graph is* cyclically k-edge-connected *if every edge cut such that at least two of the resulting components contain a circuit has size at least k.*

We sometimes omit the word "edge" as we do not use vertex connectivity. Obviously k-connectivity implies cyclic k-connectivity. On the other hand there are cubic graphs with arbitrarily large cyclic connectivity. A simple upper bound on the cyclic connectivity is the *girth* of the graph (i.e., the length of its shortest circuit).[2] It is easy to see that if G is cubic and cyclically 4-edge-connected then the only 3-cuts it contains are the *trivial* ones (i.e., around vertices). The following observation shows that the cyclic connectivity behaves almost like the normal connectivity:

Observation 1.3. *Let G be a cyclically 4-edge-connected cubic graph. Let G' be a graph created from G by removing two non-adjacent edges. Then G' is 2-edge-connected.*

[1] A circuit in a matroid is a minimal dependent set and in the case of graphical matroid it corresponds to a subgraph isomorphic to C_k for some k. For more details see, e.g., Oxley [2006].

[2] We just take the cut around the given circuit. This works always except a few small exceptions – namely K_4, $K_{3,3}$ and three parallel edges for cubic graphs – which do not have two disjoint circuits and hence they do not have a well defined cyclic connectivity.

Proof. Because G is cyclically 4-edge-connected it is also 3-edge-connected (any cut smaller than three in a cubic graph is non-trivial). Hence G' is connected. It remains to show that it is bridgeless.

We proceed by contradiction. Let b be a bridge of G' and e and f the two edges removed from G. Because b is a cut in G', the set $C = \{b, e, f\}$ is a 3-cut in G. The graph G is cyclically 4-edge-connected so C must be a trivial cut (i.e., edges around one vertex). This is a contradiction with e and f being non-adjacent. □

The first objects we want to study are double covers:

Definition 1.4 (Double Cover)**.** *Let G be a graph. A multiset of circuts (cycles, respectively) $\mathcal{C}$ is a* circuit (cycle, respectively) double cover *if every edge of G is contained in exactly two elements of $\mathcal{C}$. It is a k-cycle double cover if $|\mathcal{C}| \leq k$. We forbid double covers to contain the empty cycles. We denote $\nu(G)$ the number of circuit double covers of the graph G.*

This definition also deserves a little bit of explanation. Double cover is usually defined as a family $\{C_i\}_{i \in I}$ not a multiset. This is equivalent for the existential questions but not for counting. Even if we fix I to be integers one up to some suitable k – i.e., make the family a k-tuple – we can still create new double covers by permuting the elements of this tuple.

The difference is not too big (at most the multiplicative factor $k!$) for k-cycle double covers if k is fixed but it is crucial for circuit double covers as the number of circuits usually grows with the size of the graph. The exclusion of empty cycles is required to prevent generation infinitely many double covers by repeatedly adding the empty cycle again and again.

The other structure we are interested in are group flows. They are similar to the well-known flows in networks but there is no source and sink and instead of real numbers we use a general abelian[3] group.

Definition 1.5 (Group Flow)**.** *Let $G = (V, E)$ be a directed graph and let Γ be an abelian group. A* flow *in G is a function $\varphi\colon E \to \Gamma$ such that*

$$\sum_{e \in \delta^+(v)} \varphi(e) = \sum_{e \in \delta^-(v)} \varphi(e)$$

for every vertex $v \in V$ where $\delta^-(v)$ ($\delta^+(v)$, respectively) is set of all edges going out of (into, respectively) vertex v. We say a flow is nowhere-zero *if it does not use value 0 at any edge.*

Although this definition is usually used with finite groups, there is one interesting case of an infinite group:

Definition 1.6 (Integer Flow)**.** *Let $G = (V, E)$ be a directed graph. A $\mathbb{Z}$-flow φ is a k-flow if $|\varphi(e)| < k$ for all edges $e \in E$.*

The inequality in the definition is strict so that Theorem 1.30 of Tutte can be stated without annoying ± 1. We also recall a useful theorem about disjoint spanning trees. It is a trivial consequence of Nash-Williams theorem which was independently proven by Tutte [1961] and Nash-Williams [1961].

[3] We require the group to be abelian because there is no predefined order of the edges around each vertex. There exists a non-abelian version which works with graphs on orientable surfaces. For details see, e.g., DeVos [2000] or Goodall et al. [2018].

Theorem 1.7. *Let G be a $2k$-edge-connected graph. Then G has k disjoint spanning trees.*

Corollary 1.8. *Let G be a 3-edge-connected graph. Then G has three spanning trees such that every edge is in at most two of them.*

1.2 Snarks

As we will see in the following sections, most of the conjectures about double covers and group flows can be reduced to cubic graphs. Moreover they do not hold for graphs with bridges for trivial reasons and they usually do hold for cubic graphs which are 3-edge-colorable for reasons only slightly less trivial. This motivates the definition of snarks as the hard cases:

Definition 1.9 (Snark)**.** *A bridgeless cubic graph is a* snark *if it is not 3-edge-colorable.*

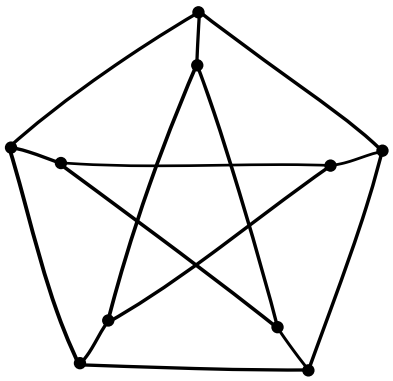

Figure 1.1: The Petersen graph

Also in many cases, it can be shown that the hypothetical minimal counterexample does not contain an edge cut of size two nor three. Hence some authors also require cyclic 4-edge-connectivity or girth at least five to exclude more trivial cases. Both of these requirements prevent generating new snarks by expanding vertices into triangles. The name "snarks" was first used by Gardner [1976] and it is a reference to the poem "The Hunting of the Snark" by Lewis Carroll.

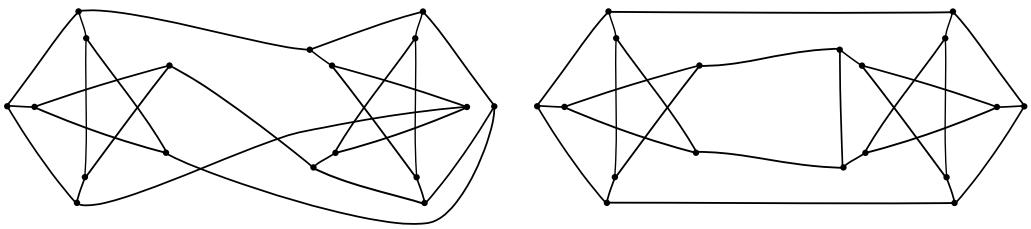

Figure 1.2: Blanuša snarks

The smallest snark is the famous Petersen graph (Figure 1.1, constructed by Petersen [1898] as the smallest not 3-edge-colorable graph) which has 10 vertices. It is cyclically 4-edge-connected and it has girth five. There is also a conjecture due to Tutte that every snark contains a subdivision of the Petersen graph (note

Table 1.3: The number of snarks of given sizes

Vertices	Girth ≥ 4	Girth ≥ 5
10	1	1
18	2	2
20	6	6
22	31	20
24	155	38
26	1,297	280
28	12,517	2,900
30	139,854	28,399
32	1,764,950	293,059
34	25,286,953	3,833,587
36	404,899,916	60,167,732

that for cubic graphs subdivisions and minors are the same). This conjecture is resolved positively by Theorem 1.41 but its proof is still not fully published.

Conjecture 1.10 (Tutte [1967])**.** *Every snark contains a subdivision of the Petersen graph.*

The next snarks with girth at least four are two Blanuša snarks both on 18 vertices (Figure 1.2, Blanuša [1946]). They can be obtained as a dot product (Definition 1.16 below) of two copies of the Petersen graph. There are two of them because up to isomorphism there are only two ways how to choose non-adjacent edges in the Petersen graph.

Then there are cyclically 4-connected snarks of every even size starting with 20 (Corollary 1.18). All the small snarks with girth at least four were enumerated by Brinkmann et al. [2013]. Their numbers are shown in Table 1.3 and they are available at `https://hog.grinvin.org`. A family of snarks which will be of particular interest are Flower snarks (defined by Isaacs [1975], for small examples see Figure 1.4):

Definition 1.11 (Flower Snarks)**.** *Flower snark J_k can be constructed in the following way:*

1. *Start with cycle C_k.*
2. *To each vertex of the cycle attach a copy of $K_{1,3}$ by identifying the vertex of the cycle and one of the degree-one vertices of $K_{1,3}$. Denote the remaining degree-one vertices of $K_{1,3}$ v_i and w_i.*
3. *Add $2k$ edges to create cycle $v_1, v_2, \ldots, v_k, w_1, w_2, \ldots, w_k$.*

Obviously J_k is a bridgeless cubic graph with $4k$ vertices and $6k$ edges. It is easy to see that J_3 is the Petersen graph with one vertex replaced by a triangle. Because it contains a triangle it is not considered to be a snark by the most definitions (but it is for us). It is also known as Tietze's graph due to Tietze [1910] who constructed it to show that a graph embedded into Möbius strip may require six colors to color its faces. Flower "snarks" J_k for even k are 3-edge-colorable hence not snarks. For odd $k \geq 5$ they are indeed snarks:

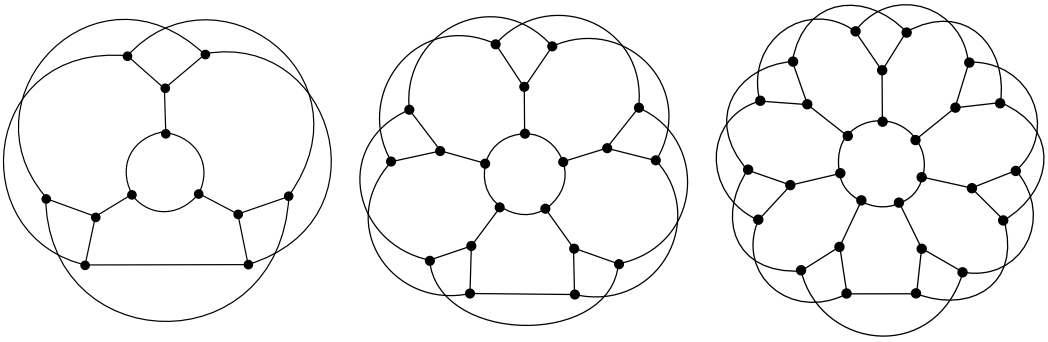

Figure 1.4: Flower snarks J_3, J_5 and J_7

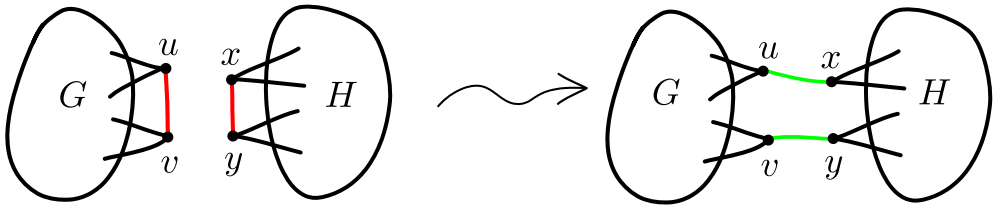

Figure 1.5: 2-sum

Theorem 1.12 (Isaacs [1975]). *Flower snarks J_k for odd $k \geq 5$ are snarks, J_5 has cyclic connectivity 5 and the others have cyclic connectivity 6.*

We present a few ways to construct snarks from smaller snarks. The simplest constructions are a 2-sum (this operation was first described by Kochol [2002] and he used it reduce 3-flow conjecture to 5-edge-connected case) and a 3-sum but they create small cuts.

Definition 1.13 (2-sum, Kochol [2002]). *Let G and H be bridgeless cubic graphs. Let $uv \in E(G)$ and $xy \in E(H)$ be two of their edges. Then the 2-sum of G and H is the following graph (see Figure 1.5):*

$$(G - uv) \uplus (H - xy) + \{ux, vy\}.$$

Definition 1.14 (3-sum). *Let G and H be bridgeless cubic graphs. Let $u \in V(G)$ and $v \in V(H)$ be two of their vertices. Let u_1, u_2, u_3 be the neighbors of u and v_1, v_2, v_3 be the neighbors of v. Then the 3-sum of G and H is the following graph (see Figure 1.6):*

$$(G - u) \uplus (H - v) + \{u_1v_1, u_2v_2, u_3v_3\}.$$

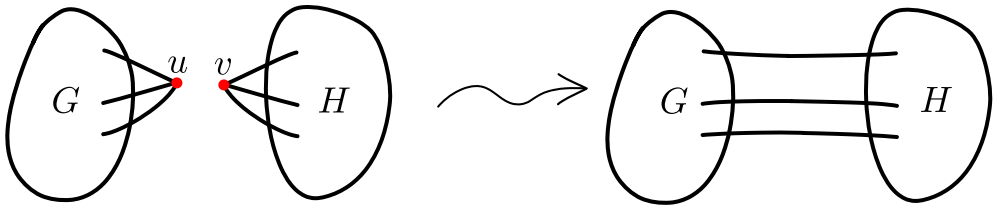

Figure 1.6: 3-sum

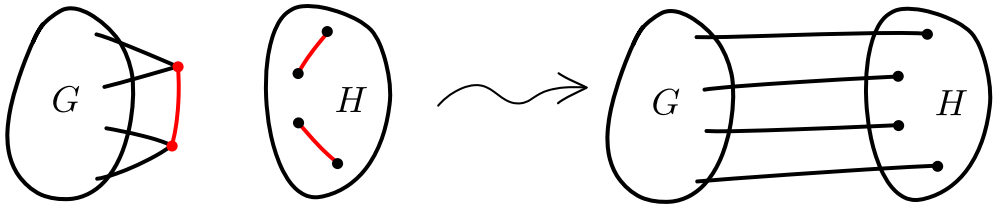

Figure 1.7: Dot product

Observation 1.15. *Let G_1 and G_2 be bridgeless cubic graphs and G their 2-sum or 3-sum. If at least one of G_1 and G_2 is a snark then G is a snark.*

Proof. Using Observation 1.37, we can work with nowhere-zero $\mathbb{Z}_2^2$-flows instead of 3-edge-colorings. Hence we can just contract the other part of G (and suppress the new vertex of degree two in the case of 2-cut) and get nowhere-zero $\mathbb{Z}_2^2$-flow on the remaining part which is a contradiction with it being a snark. □

A better construction which can create cyclically 4-connected graphs is the dot product. The dot product was introduced independently by Adelson-Velskij and Titov [1974] and Isaacs [1975]:

Definition 1.16 (Dot Product)**.** *Let G and H be cubic graphs. Let e be an edge of G, u, v the neighbours of one end of e and x, y the neighbours of the other end of e. Let ab and cd be edges of H. The graph $G \cdot H$ is created from the disjoint union of G and H by deleting edges ab, cd, deleting edge e with its endpoints and adding edges au, bv, cx and dy (see Figure 1.7).*

Theorem 1.17 (Isaacs [1975])**.** *Let G and H be snarks. Then $G \cdot H$ is also a snark. If both G and H are cyclically 4-edge-connected and if the vertices a, b, c, d are all different, then $G \cdot H$ is also cyclically 4-edge-connected.*

By taking a few small snarks (e.g., found by Brinkmann et al. [2013]) and doing dot product with the Petersen graph repeatedly, we get the following corollary:

Corollary 1.18. *For every even $n \geq 20$ there exists a cyclically 4-connected snark with n vertices.*

The last construction we describe was introduced by Kochol [1996a,b] and it is called a superposition. Its advantage is that it can be used to construct cyclically 5 or 6-connected snarks. The basic idea is to take a snark and replace each vertex and edge with some other graph. For this we need a definition of a pole.

Definition 1.19 (Supervertex and Superedge)**.** *A $(k_1, k_2, \ldots, k_t)$-*pole*, denoted $M(V, E, S_1, \ldots, S_k)$, is a graph with degrees 3 and 1 such that $S_1, \ldots, S_k$ is a partition of the edges incident with the degree 1 vertices and $|S_i| = k_i$ for all i. Each S_i is called a* connector*.*

A (k_1, k_2, k_3)-pole is called a supervertex *and a (k_1, k_2)-pole is called a* superedge*. A superedge $M(V, E, S_1, S_2)$ is* proper *if $\sum_{e \in S_1} \varphi(e) \neq 0$ for every 3-edge-coloring $\varphi : E \to \mathbb{Z}_2^2 \setminus \{0\}$.*

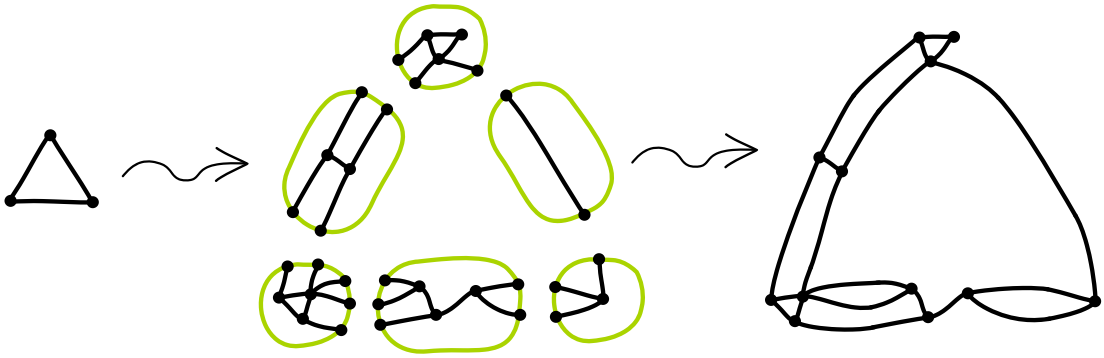

Figure 1.8: An example of a superposition

Definition 1.20 (Superposition)**.** *Let G be a cubic graph, $\mathcal{V} = \{X_v : v \in V(G)\}$ be a set of supervertices, $\mathcal{E} = \{F_e : e \in E(G)\}$ be a set of superedges and $\omega : V(G) \times E(G) \to \{1, 2, 3\}$ and $\varepsilon : V(G) \times E(G) \to \{1, 2\}$ be maps which map vertex-edge incidences to connectors. Then their* superposition *is a graph H obtained by replacing vertices of G by X_v, edges of G by F_e, identifying the degree 1 vertices of the connector $\omega(v, e)$ with vertices of $\varepsilon(v, e)$ and suppressing the created vertices of degree 2 (for an example see Figure 1.8).*

Theorem 1.21 (Kochol [1996b])**.** *Let G be a graph created by a superposition from graph H. If H is a snark and all the used superedges are proper then G is a snark.*

We also note that it is possible to construct snarks by a technique called *voltage graphs*. But this technique by itself does not guarantee that the result is a snark. More details about voltage graphs can be found in Chapter 7. We conclude this section with a few results about snarks with high cyclic connectivity.

Theorem 1.22 (Kochol [1996a])**.** *There is an infinite family of cyclically 6-edge-connected snarks.*

Theorem 1.23 (Kochol [1996b])**.** *For every fixed $k \in \mathbb{N}$ there is an infinite family of cyclically 5-edge-connected snarks with girth $\geq k$.*

On the other hand every cubic graph with a Hamiltonian cycle is 3-edge-colorable and there is a conjecture that every sufficiently connected cubic graph is Hamiltonian. The Coxeter graph [Coxeter, 1983] shows that k must be at least 8 and there is no known graph showing that $k > 8$.

Conjecture 1.24 (Thomassen [1996])**.** *There exists $k \in \mathbb{N}$ such that every cyclically k-connected cubic graph is Hamiltonian.*

1.3 Flows

Throughout this section G is a directed graph (but the edge orientation does not matter) and Γ is an abelian group unless noted otherwise. We start with an equivalent alternative definition of the group flows which uses surpluses at vertices:

Definition 1.25 (Surplus)**.** *Let $\varphi : E(G) \to \Gamma$ be any mapping. We define* surplus *of φ for all vertices $v \in V(G)$:*

$$\partial\varphi(v) = \sum_{e \in \delta^+(v)} \varphi(e) - \sum_{e \in \delta^-(v)} \varphi(e).$$

Observation 1.26 (Equivalent Definition of Γ-flow)**.** *A mapping $\varphi : E(G) \to \Gamma$ is a Γ-flow if and only if $\partial\varphi \equiv 0$.*

Proof. Obvious. □

Observation 1.27. *For every mapping $\varphi : E(G) \to \Gamma$ it holds*

$$\sum_{v \in V(G)} \partial\varphi(v) = 0.$$

Proof. Every edge e contributes $\varphi(e)$ to surplus of one of its endpoints and $-\varphi(e)$ to the other one. Hence its overall contribution is zero. □

Observation 1.28. *Let T be a spanning tree of G. Then every mapping $\varphi : E(G) \setminus E(T) \to \Gamma$ can be extended into a flow $E(G) \to \Gamma$.*

Proof. Choose an arbitrary root of the spanning tree. Work from the leaves up to the root and always assign the edge such a value to make surplus of its lower vertex zero. This makes it a flow in all vertices but the root. But the surplus of the root is also zero as the sum of all surpluses is zero. □

Tutte [1954] started the study of nowhere-zero flows by observing, that a plane digraph G has a nowhere-zero flow in $\mathbb{Z}_k$ if and only if its plane dual G^* is k-vertex-colorable (we do not consider orientation of the edges for the coloring). He also showed several nice properties of the group flows – namely their existence depends only on the size of the group and not on its structure and also their existence is monotone in the size of the group.

Theorem 1.29 (Tutte [1954])**.** *Let Γ be an abelian group with k elements. Then a directed graph has a nowhere-zero Γ-flow if and only if it has a nowhere-zero $\mathbb{Z}_k$-flow.*

Theorem 1.30 (Tutte [1954])**.** *Then a directed graph has a nowhere-zero $\mathbb{Z}_k$-flow if and only if it has a nowhere-zero k-flow.*

Now we start examining the group flows for small groups. Obviously a graph has nowhere-zero 1-flow only if it does not have any edges. To have a nowhere-zero 2-flow, all degrees of the vertices must be even so the graph must be a cycle. The situation is much more interesting for 3-flows:

Observation 1.31. *A cubic graph has a nowhere-zero $\mathbb{Z}_3$-flow if and only if it is bipartite.*

Proof. The if part first. We orient the edges from one partition to the other and assign one to all of them. This is a valid nowhere-zero $\mathbb{Z}_3$-flow. For the only-if part see that we can change directions of the edge so the flow on all of them is one. But then each vertex is either a source or a sink (i.e., all the incident edges are oriented out of the given vertex or all the edges are oriented into it). So sources are one partition and sinks the other one. □

Conjecture 1.32 (Tutte [1954]). *Every 4-edge-connected graph has a nowhere-zero 3-flow.*

The 3-flow conjecture is a strengthening of Grötzsch's theorem which is by duality equivalent to an assertion that every 4-edge-connected planar graph has a nowhere-zero 3-flow.

Theorem 1.33 (Grötzsch [1959]). *Every planar triangle-free graph is 3-vertex-colorable.*

A weaker version which asked if k-connectivity for some fixed k is enough to ensure existence of a nowhere-zero 3-flow was proposed by Jaeger [1979]. This version was solved by Thomassen and further improved by Lovász, Thomassen, Wu and Zhang.

Theorem 1.34 (Thomassen [2012]). *Every 8-edge-connected graph has a nowhere-zero 3-flow.*

Theorem 1.35 (Lovász et al. [2013]). *Every 6-edge-connected graph has a nowhere-zero 3-flow.*

It is also known that the conjecture is true if we replace 3-flow with a 4-flow:

Observation 1.36. *Every 4-edge-connected graph has a nowhere-zero 4-flow.*

Proof. Because G is 4-edge-connected, it has two disjoint spanning trees T_1 and T_2 due to Theorem 1.7. Hence we can construct a nowhere-zero $\mathbb{Z}_2^2$-flow φ by setting φ to 1 in the first coordinate for all edges except T_1 and to 1 in second coordinate out of T_2. Due to Observation 1.28 this partial mapping can be extended to a flow. □

One of important properties of 4-flows is that for cubic graphs their existence is equivalent to 3-colorings:

Observation 1.37. *A cubic graph has nowhere-zero $\mathbb{Z}_2^2$-flow if and only if it is 3-edge-colorable.*

Proof. There are three non-zero values in $\mathbb{Z}_2^2$ – namely $(1,0)$, $(0,1)$ and $(1,1)$. The equation $a+b+c=0$ is satisfied in $\mathbb{Z}_2^2$ if and only if $\{a,b,c\} = \{(1,0),(0,1),(1,1)\}$. Hence we can use any bijection from colors to $\{(1,0),(0,1),(1,1)\}$ to obtain a nowhere-zero $\mathbb{Z}_2^2$-flow from a 3-coloring and vice versa a nowhere-zero $\mathbb{Z}_2^2$-flow is a 3-coloring. □

This immediately gives very simple proofs of the known facts that a cubic graph with a bridge is not 3-colorable and that edges of a 2-cut must have the same color. Both due to the fact that flow on every cut is zero. So a large class of cubic graphs has nowhere-zero 4-flow but other do not. Specially the Petersen graph does not as it is not 3-colorable.

Corollary 1.38. *The Petersen graph has no nowhere-zero $\mathbb{Z}_4$-flow.*

This motivates the next Tutte's conjecture:

Conjecture 1.39 (Tutte [1954]). *Every bridgeless graph without the Petersen minor has nowhere-zero 4-flow.*

The 4-flow conjecture cannot be reduced to cubic graphs. At least not by the usual vertex splitting lemma because it can introduce new minors:

Observation 1.40 (Vertex Splitting Lemma, Fleischner [1992]). *Let G be a 2-edge-connected graph and let v be its vertex of degree at least 4. Let uv, w_1v and w_2v be three edges incident with v. Then at least one of graphs $G-uv-w_1v+uw_1$ and $G-uv-w_2v+uw_2$ is 2-edge-connected.*

But it can be viewed as a generalization of the Four color theorem [Appel and Haken, 1977, Appel et al., 1977]. The Four color theorem is equivalent to a statement that every bridgeless cubic planar graph has nowhere-zero 4-flow (shown by Tait [1880]). Hence even the restriction of the 4-flow conjecture to cubic graphs is a strengthening of the Four color theorem. Thomas [1999] announced that he together with Neil Robertson, Daniel P. Sanders and Paul Seymour solved this restricted version but the proof is still not fully published today (2021).

Theorem 1.41 (Thomas [1999], proof not yet fully published). *Every bridgeless cubic graph without the Petersen minor has nowhere-zero 4-flow.*

We finish with 5-flows. The existence of a nowhere zero 5-flow is probably the biggest open question in this area:

Conjecture 1.42 (Tutte [1954]). *Every bridgeless graph has a nowhere-zero 5-flow.*

This conjecture can easily be reduced to cubic graphs and hence to snarks. There first step towards this conjecture was made by Jaeger:

Theorem 1.43 (Jaeger [1979]). *Every bridgeless graph has a nowhere-zero 8-flow.*

The proof constructs a $\mathbb{Z}_2^3$-flow using the fact that every 3-connected graph has three spanning trees such that every edge is in at most two of them (Corollary 1.8). Seymour improved the bound to 6 by constructing a $\mathbb{Z}_3 \times \mathbb{Z}_2$-flow. To do this he found a family of circuits whose contraction makes the graph "connected" enough to have a 3-flow.

Theorem 1.44 (Seymour [1981]). *Every bridgeless graph has a nowhere-zero 6-flow.*

Note that there is no direct proof of existence of a nowhere-zero 7-flow as all the proofs depend on constructing the final flow from several flows in smaller groups. This is also one of the reasons why the 5-flow conjecture still resists solving. But there are some facts we know about the hypothetical minimal counterexample to the 5-flow conjecture:

1. It is a snark.

2. It is cyclically 6-edge-connected [Kochol, 2004].

3. It has girth at least 12.[4]

There is also a strengthening of nowhere-zero flows called group connectivity. It was introduced by Jaeger et al. [1992] and we explore more it in Chapter 2.

1.4 Double Covers

The main open question in this area is the following conjecture due to Szekeres [1973], independently Seymour [1979] and in slightly modified but equivalent version by Itai and Rodeh [1978]:

Conjecture 1.45 (Circuit Double Conjecture, Szekeres [1973])**.** *Every bridgeless graph has a circuit double cover.*

Like the 5-flow conjecture, this and other conjectures about double covers can easily be reduced to cubic graphs and snarks. We say that a double cover is *orientable* if it is possible to assign a direction to each of its circuits such that every edge is covered once with the same direction and once with the opposite direction. This leads to a natural strengthening of the CDC[5] conjecture:

Conjecture 1.46 (Oriented CDC)**.** *Every graph has an orientable cycle double cover.*

There is a tight relationship between double covers with a few cycles and group flows. Note that being a cycle is equivalent to having a 2-cycle double cover.

Observation 1.47. *A graph has a nowhere-zero 2-flow if and only if it is a cycle.*

Proof. Obvious. □

Theorem 1.48 (Tutte [1949])**.** *A graph has a nowhere-zero 3-flow if and only if it has an orientable 3-cycle double cover.*

Observation 1.49. *A graph has an (orientable) k-cycle double cover if and only if it has a $\mathbb{Z}^k$-flow φ such that for every edge e the value $\varphi(e)$ contains exactly one 1, one -1 and the rest are zeros, resp. two are non-zeros from the set $\{\pm 1\}$ and rest are zeros in the non-orientable case.*

Proof. Obvious. □

The next observation was originally published in pieces in multiple papers. In this form it was formulated in Chapter 3.1 of Zhang [1997]. But its proof is simple and nicely illustrates working with double covers so we present it here anyway.

Observation 1.50. *The following statements are equivalent for a graph G:*

[4] Kochol [2010] proved 11. This was improved by Hušek et al. [2016] which was presented at Bordeaux Graph Workshop but it still remains unpublished although the proof uses the same computer aided technique as Kochol. It is also worth noting that Kochol's approach fails when trying to exclude C_{12} as subgraph.

[5] We say CDC instead of cycle double cover or circuit double cover. It should be obvious from context which one we mean (usually cycle up to Chapter 4 and circuit from there on).

1. *Graph G has a nowhere-zero 4-flow.*
2. *Graph G has a 3-cycle double cover.*
3. *Graph G has a 4-cycle double cover.*
4. *Graph G has an orientable 4-cycle double cover.*

Proof. We use a $\mathbb{Z}_2^2$-flow instead of a 4-flow. We extend the $\mathbb{Z}_2^2$-flow φ into $\mathbb{Z}_2^3$-flow φ' via formula $(x, y) \mapsto (x, y, x + y)$ and use Observation 1.49 to obtain the equivalence of (1) and (2). Both item (2) and (4) trivially imply (3). It remains to show (3) $\Rightarrow$ (2) and (2) $\Rightarrow$ (4).

(3) $\Rightarrow$ (2): Let C_1 up to C_4 be the cycles of the 4-CDC. Then $C_1 \triangle C_2$, $C_1 \triangle C_3$, $C_1 \triangle C_4$ is also a CDC (where $\triangle$ is the symmetric difference).

(2) $\Rightarrow$ (4): Let C_1, C_2, C_3 be the cycles of the 3-CDC. Define three 2-flows f_i such that f_i is non-zero along the cycle C_i and zero elsewhere. Then we use Observation 1.49 on the flow φ:

$$\varphi = \left(\frac{f_1 + f_2 + f_3}{2}, \frac{f_1 - f_2 - f_3}{2}, \frac{-f_1 + f_2 - f_3}{2}, \frac{-f_1 - f_2 + f_3}{2} \right). \qquad \square$$

A direct consequence is that the Petersen graph does not have a 4-cycle double cover.

Corollary 1.51. *The Petersen graph has no 4-cycle double cover.*

On the other hand it has an orientable 5-cycle double cover and there is no known graph for which 5 cycles does not suffice:

Conjecture 1.52 (5-CDC)**.** *Every bridgeless graph has 5-cycle double cover.*

Conjecture 1.53 (5-OCDC)**.** *Every graph has an orientable 5-cycle double cover.*

The relationship between flows and double covers also exists for larger k:

Observation 1.54. *Every graph with an orientable k-cycle double cover has a nowhere-zero k-flow.*

Proof. Let $f_i : E(G) \to \{\pm 1\}$ be the cycles of the double cover. Define flow $\varphi = \sum_{i=1}^{k} i f_i$. Then φ is a nowhere zero k-flow. $\square$

Corollary 1.55. *Orientable 5-cycle double conjecture implies 5-flow conjecture.*

The opposite implication is known for $k \leq 4$ as shown above but it is still an open problem for 5 and 6. We study a possible direction for proving this in Chapter 3. For a non-orientable CDC we can use the same construction but the result will be a $2k$-flow.

There are also strengthenings of CDC conjecture going in other directions. For example it is still open whether any circuit of a graph can be extended into a circuit double cover. On the other hand it is known to be false if we replace circuits with cycles.

Conjecture 1.56 (Strong CDC)**.** *Let G be a bridgeless cubic graph and let C be a circuit of G. Then there exists a circuit double cover of G containing C.*

1.4.1 Generalized Cycle Covers

It is natural to ask why to cover each edge twice and not some other number of times. A summary of known results is in Table 1.9 but to examine this we need the following technical definition:

Definition 1.57 (General Cycle Cover)**.** *Let $M \subseteq \mathbb{Z}^+$ be a non-empty set. An* (n, M)-cover *of graph G is a system of at most n cycles such that each edge of G is covered m times for some $m \in M$.*

We write (n, m)-cover instead of $(n, \{m\})$-cover. Hence double covers are $(\omega, 2)$-covers. It is easy to see that any $(n, \{1, 2\})$-cover can be extended into a double cover. More generally every cover can be modified so all the edges are covered the same number of times:

Observation 1.58. *Let $\mathcal{C}$ be a (n, M)-cover of G. Then there exists a $(2^n - 1, 2^{n-1})$-cover C' of G.*

Proof. Let $\mathcal{C} = (C_1, \ldots, C_n)$, treat C_i as $\mathbb{Z}_2$-flows and define

$$\mathcal{C}' = \left(\sum_{i \in S} C_i : \emptyset \neq S \subseteq \{1, 2, \ldots, n\} \right).$$

Obviously $\mathcal{C}'$ has $2^n - 1$ cycles. It covers each edge 2^{n-1} times because every fixed set Y has $2^{|X|-1}$ intersections of odd size with all the subsets of any other fixed set X given $X \cap Y \neq \emptyset$. □

Table 1.9: Overview of (n, m)-covers

n	$m = 2$	$m = 4$	$m = 6$
2	Eulerian	–	–
3	4-flow	–	–
4	4-flow	Eulerian	–
5	open (5-CDC)	5-postman set	–
6	open	open (Berge-Fulkerson)	Eulerian
7	open	yes	7-postman set
8	open	yes	open
9	open	yes	open
10	open	yes	yes [Fan, 1992]

Using 8-flow theorem (Theorem 1.43) we obtain a $(3, \{1, 2, 3\})$-cover and by the previous observation a $(7, 4)$-cover. This was first proved by Bermond, Jackson and Jaeger:

Theorem 1.59 (Bermond et al. [1983])**.** *Every bridgeless graph has a $(7, 4)$-cover.*

The complement of a spanning cycle in a cubic graph is a perfect matching. This lead from cycle covers to Berge-Fulkerson conjecture about perfect matchings:

Conjecture 1.60 (Berge-Fulkerson, Fulkerson [1971])**.** *Every cubic bridgeless graphs has six perfect matchings such that every edge is in exactly two of them.*

Conjecture 1.61 (Reformulation of Berge-Fulkerson Conjecture, Jaeger [1979])**.** *Every bridgeless graph has a* $(6, 4)$*-cover.*

Definition 1.62 (Postman Set)**.** *A set of edges* $J \subset E(G)$ *is a* postman set *if* $E(G) \setminus J$ *is a cycle. A* k-postman set *is a partition of edges of a graph into exactly* k *postman sets.*[6]

Observation 1.63. *A graph has* k*-postman set if and only if it has* $(k, k-1)$*-cover.*

1.4.2 Double Covers and Surfaces

There is also a connection between double covers and embeddings of graphs into surfaces. For more details about graph embeddings see, e.g., Mohar and Thomassen [2001].

For us a surface is a compact 2-manifold (i.e., it is locally homeomorphic to $\mathbb{R}^2$) without boundary – i.e., sphere, projective plane, torus, etc. It is easy to see that every graph can be embedded without crossing edges into some surface – just start with a plane and add a handle (also known as ear) every time you need to avoid crossing edges.

For our purposes we need a stronger notion of embedding which ensures that every face is an disk and a walk around the face is a circuit in the graph. Such embeddings are called circular. When we say just embedding we always mean a circular one unless noted otherwise.

Definition 1.64 (2-cell Embedding)**.** *Let* Σ *be a 2-manifold and let* G *be a graph. The embedding of* G *into* Σ *is a* 2-cell embedding *if the interior of every face is an open disc.*

Definition 1.65 (Circular Embedding)**.** *A 2-cell embedding a* circular *embedding if the boundary of each face is a circuit.*

Obviously given a circular embedding the collection of its facial walks is a circuit double cover. Moreover it is an orientable CDC if and only if the surface was orientable. Vice versa given a circuit double cover we can use the graph as a skeleton and glue a disk along every circuit of the CDC. The caveat is that this might not be a surface.

Consider a vertex of degree four covered in such a way that two of its edges are covered by cycles a and b and the other two edges by c and d. Then the created "surface" at the vertex looks like two planes with a single point identified – such objects are usually called pseudo-surfaces. But for such a thing to happen a vertex of degree four or more is required. If the graph is cubic, the result is always a surface.

Hence the following conjecture restricted to cubic graphs is equivalent to the circuit double cover conjecture:

Conjecture 1.66 (Haggard [1977], Little and Ringeisen [1978])**.** *Every 2-vertex-connected graph has circular 2-cell embedding.*

[6] The empty set might be repeated.

When embedding in a sphere (or equivalently in a plane), any closed curve on the surface which does not pass through vertices and intersects each edge in at most one point is either disjoint with the embedded graph or corresponds to an edge cut. The situation is more complicated for other surfaces. On all the other surfaces there are closed curves which do not split the surface into two parts. Such curves are called *noncontractible*. It is an important property of an embedding how many edges must every noncontractible curve intersect. This number is called representativity or face-width.

Definition 1.67 (Representativity)**.** *Let* Γ *be an embedding of graph* G *in a surface which is not a sphere. The* representativity *of this embedding is the minimal number of edges the embedding any closed noncotractible curve must intersect given it does not intersect vertices.*

We again conclude this section with the known facts about the hypothetical minimal counterexample G to circuit double cover conjecture (Chapter 7.2 of Zhang [1997]):

1. G is a snark.
2. G is cyclically 4-edge-connected.
3. G contains a subdivision of the Petersen graph.
4. G has girth at least 10.
5. If G' is created from G by either addition or removal of a single edge then G' does not have 4-flow.

2. Group Connectivity

This chapter is based on work which was published as journal paper Hušek et al. [2019]. Han et al. [2020] extended our results to 3-edge-connected graphs. We summarize their contribution in Section 2.5.

Jaeger introduced a variant of nowhere-zero flows called *group connectivity*:

Definition 2.1 (Jaeger et al. [1992])**.** *A directed graph $G = (V, E)$ is* Γ-connected *if for every mapping $h\colon E \to \Gamma$ there is a Γ-flow φ on G that satisfies $\varphi(e) \neq h(e)$ for every edge $e \in E$.*

As we may choose the "forbidden values" $h \equiv 0$, every Γ-connected digraph has a nowhere-zero Γ-flow; the converse is false, however. While the notion of group connectivity is stronger than the existence of nowhere-zero flows, it is also more versatile, in particular the notion lends itself more easily to proofs by induction. This is a consequence of an alternative definition of group connectivity: instead of looking for a flow, we may check existence of a mapping $E \to \Gamma$ that has prespecified surplus at each vertex.

It is easy to see that both the existence of a nowhere-zero Γ-flow and Γ-connectivity do not change when we reverse the orientation of an edge of the digraph (we only need to change the corresponding flow value from x to $-x$). Thus, we will say that an undirected graph G has a nowhere-zero Γ-flow (is Γ-connected) if some (equivalently every) orientation of G has a nowhere-zero Γ-flow (is Γ-connected). Also, using the definition of group connectivity working with vertex surpluses, we observe that group connectivity is monotone with respect to edge addition – if G is Γ-connected then $G + e$ is Γ-connected for any edge e.

Some results on nowhere-zero flows extend to the stronger notion of group connectivity. A celebrated recent example of this is the solution to the Jaeger's conjecture by Lovász et al. [2013], but there are many more. Thus, it is worthwhile to understand the properties of group connectivity in more detail.

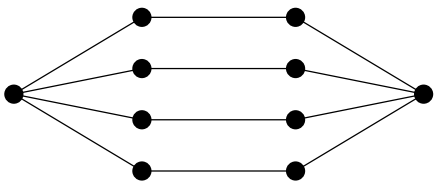

Figure 2.1: A graph which is $\mathbb{Z}_5$ but not $\mathbb{Z}_6$-connected

However, some nice properties of group-valued flows are not shared by group connectivity. In particular Jaeger et al. [1992] showed that there is a graph (Figure 2.1) that is $\mathbb{Z}_5$-connected, but not $\mathbb{Z}_6$-connected. On the other hand, Theorem 1.30 indicates that a graph G admitting a nowhere zero flow in $\mathbb{Z}_5$ also has a nowhere zero flow in $\mathbb{Z}_6$.

An analogy of Theorem 1.29 is more subtle. Indeed, in Section 3.1 of Jaeger et al. [1992] the authors mention: "... we do not know of any $\mathbb{Z}_4$-connected graph which is not $\mathbb{Z}_2 \times \mathbb{Z}_2$-connected, or vice versa. Neither can we prove that such graphs do not exist." Our main result is the resolution to this natural question.

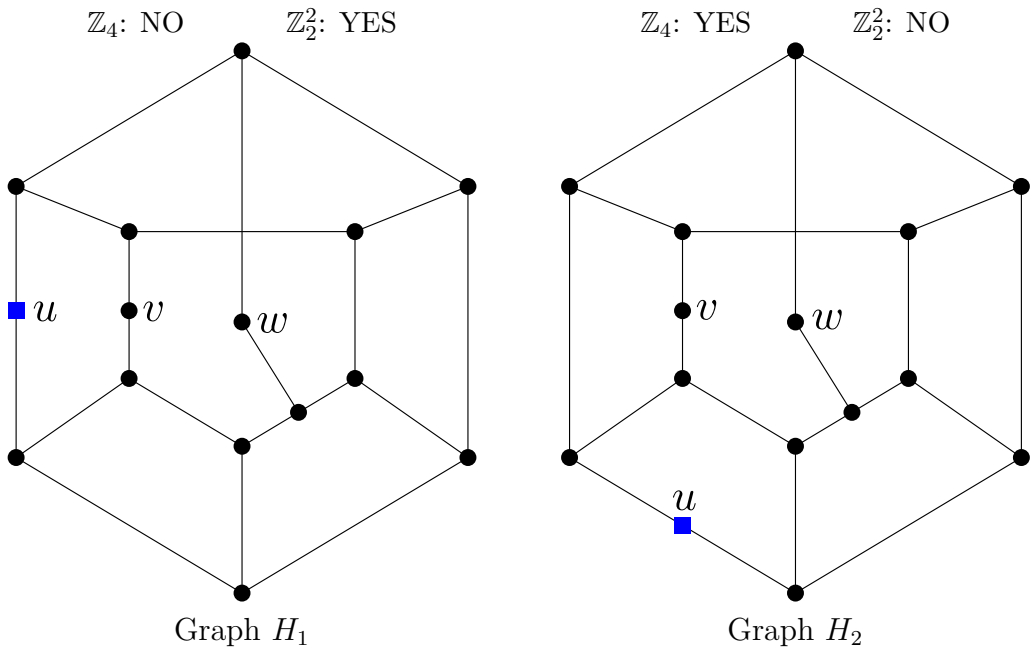

Figure 2.2: Graphs proving Theorem 2.2. Note that they are the same except the blue square vertices u.

Theorem 2.2. *Let H_1 and H_2 be the graphs shown in Figure 2.2.*

1. *Graph H_1 is $\mathbb{Z}_2^2$-connected but not $\mathbb{Z}_4$-connected.*

2. *Graph H_2 is $\mathbb{Z}_4$-connected but not $\mathbb{Z}_2^2$-connected.*

Because our result is computer aided, we do not present the proof in a classical sense. Instead we present an overview of our approach and examples of graphs proving Theorem 2.2 in the next section. In Section 2.2 we describe two algorithms we used to test group connectivity, and we add some implementation notes in Section 2.3.

The presented examples can be turned into infinite families with the same $\mathbb{Z}_4$ and $\mathbb{Z}_2^2$ group connectivity by repetitive application of the following observation:

Observation 2.3. *Let Γ_1, Γ_2 be groups of size at least 4, let G be a graph which is Γ_1-connected but not Γ_2-connected, and let v be its vertex of degree 3. Then graph $G^\triangle$ obtained by replacing v with a triangle (with each of the original edges of v incident to one vertex of the triangle) is also Γ_1-connected but is not Γ_2-connected.*

Proof. Suppose $G^\triangle$ is Γ_2-connected and consider any mapping $h : E(G) \to \Gamma_2$. We extend it to $h' : E(G^\triangle) \to \Gamma_2$ (say, by zeros). As $G^\triangle$ is Γ_2-connected, there is a flow ϕ' on $G^\triangle$ satisfying $\phi'(e) \neq h'(e)$ for every edge e. By contracting the new triangle we obtain a flow on G avoiding the values of h, a contradiction. On the other hand G is Γ_1-connected so for any forbidden mapping on $G^\triangle$ we can find a flow satisfying all the edges except the triangle. But the triangle has only 3 edges and $|\Gamma_1| \geq 4$ so we can modify the flow (by adding an appropriate flow supported on the triangle) to avoid all of the forbidden values. □

Also Li [2018] suggested to us that it is possible to make presented examples either cubic (but with a nontrivial 2-cut) or 3-edge-connected (but with a few vertices of degree 4) using the 2-sum operation (Definition 1.13).

2.1 The Conjecture and Results

When looking for graphs certifying Theorem 2.2, we only need to consider graphs that do have nowhere-zero $\mathbb{Z}_2^2$-flow (equivalently, by Theorem 1.29, nowhere-zero $\mathbb{Z}_4$-flow). It is natural to examine cubic graphs (and their subdivisions) due to the following theorem:

Theorem 2.4 (Jaeger et al. [1992])**.** *Let G be an 4-edge-connected graph. Then G is both $\mathbb{Z}_2^2$- and $\mathbb{Z}_4$-connected.*

Contrary to the usual case, however, we are not interested in snarks (cubic graphs that are not to be edge 3-colorable), as those do not have nowhere-zero $\mathbb{Z}_2^2$-flow.

We note that subdividing an edge has no effect on the existence of a nowhere-zero flow (the new edge can have the same flow value as before). It makes the group connectivity stronger – in effect, we are forbidding one more value on an edge. This suggests the following strategy:

1. pick an arbitrary / random 3-regular graph and
2. repeatedly subdivide an edge and check $\mathbb{Z}_2^2$- and $\mathbb{Z}_4$-connectivity.

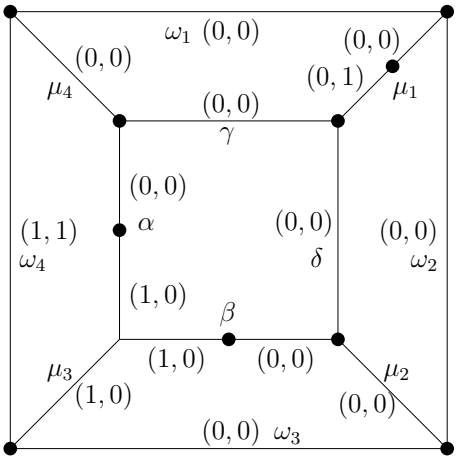

Figure 2.3: A subdivision of the cube which is $\mathbb{Z}_4$-connected but not $\mathbb{Z}_2^2$-connected with forbidden assignment for which no satisfying $\mathbb{Z}_2^2$-flow exists and names for hypothetical flow values.

This procedure yielded the graph in Figure 2.3, which appeared in the master thesis Mohelníková [2014]. This graph is $\mathbb{Z}_4$- but not $\mathbb{Z}_2^2$-connected. Later, with more effective implementation (see the next section) by Hušek, we found graphs that are $\mathbb{Z}_2^2$- but not $\mathbb{Z}_4$-connected. The smallest $\mathbb{Z}_2^2$- but not $\mathbb{Z}_4$-connected graphs we are aware of are (threefold) subdivisions of cubic graphs on 12 vertices (for an

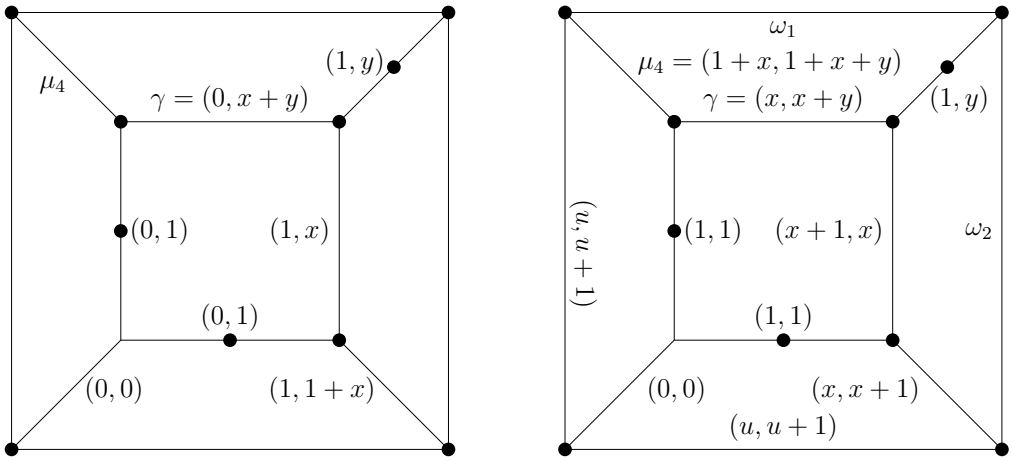

Figure 2.4: Cases $\alpha = (0,1)$ and $\alpha = (1,1)$ with fragments of hypothetical flows

example see Figure 2.2).[1] We also include a proof that graph in Figure 2.3 is not $\mathbb{Z}_2^2$-connected which is not computer-aided:

Theorem 2.5. *The subdivision of the cube in Figure 2.3 is not $\mathbb{Z}_2^2$-connected.*

Proof. We will show that for the assignment of the forbidden values in Figure 2.3 there exists no satisfying $\mathbb{Z}_2^2$-flow. First observe that values α and β are of the form $(.,1)$ which implies $\mu_3 = (.,0)$. So μ_3 is always $(0,0)$ and $\alpha = \beta$. Also $\mu_1 = (1,.)$.

Propagation of values of flow in the case $\alpha = (0,1)$ is shown in Figure 2.4, on the left. As $\mu_2 \neq (0,0)$, we have $\delta = (1,.)$. The value $x+y$ is 1 because γ is forbidden to be $(0,0)$ but this forces $\mu_4 = (0,0)$ which is also forbidden. In the case $\alpha = (1,1)$ (Figure 2.4, on the right), we again combine the forbidden values to give possible form for μ_2 and δ, and also ω_3, ω_4. In particular $\omega_3 = \omega_4 \notin \{(0,0),(1,1)\}$, so we may write $\omega_3 = (u, u+1)$. The edge γ forbids $x = y = 0$ and the edge μ_4 forbids $x = 1$, $y = 0$, so $y = 1$ and $\mu_4 = (x+1, x)$. So either $\omega_1 = (u+x+1, u+x+1)$ or $\omega_2 = (u+x, u+x)$ are $(0,0)$. Hence no satisfying flow exists. □

A proof that our example in the other direction is not $\mathbb{Z}_4$-connected is similarly easy, we omit it. We have been unable to find a proof of the positive statements: that the graphs we have found are connected with respect to the appropriate group.

2.2 Group Connectivity Testing

We fix a digraph $G = (V, E)$. We let n be the number of vertices and m the number of edges of G.

[1] We found more examples than we present here. An incomplete list of them is located at https://gitlab.kam.mff.cuni.cz/radek/group-connectivity-pub/blob/master/graph_list.py.

Notation 2.6. *We say that a flow $\varphi\colon E \to \Gamma$* satisfies *a mapping of forbidden values $h\colon E \to \Gamma$ if for every $e \in E$ it holds $h(e) \neq \varphi(e)$.*

The most straightforward way of testing whether a graph is Γ-connected, is using the definition: We can enumerate all $h\colon E \to \Gamma$ assignments of forbidden values and for each of them (try to) find a satisfying flow. Finding a satisfying flow by itself is a hard problem: A cubic graph has nowhere-zero $\mathbb{Z}_4$-flow (equivalently, $\mathbb{Z}_2^2$-flow) if and only if it has an edge 3-coloring. Testing the edge 3-colorability of cubic graphs was shown to be NP-complete by Holyer [1981].

An easy observation about the structure of forbidden assignments is:

Observation 2.7. *Let $h, h'\colon E \to \Gamma$ be assignments of the forbidden values such that $h' - h = \Delta$ is a flow. Then h is satisfied by a flow φ if and only if h' is satisfied by $\varphi + \Delta$.*

Definition 2.8. *We say that assignments of forbidden values $h, h'\colon E \to \Gamma$ are flow-equivalent, denoted $h \sim_f h'$, if and only if $h' - h$ is a flow.*

Hence we can split all assignments of the forbidden values into equivalence classes of $\sim_f$ and test the existence of a satisfying flow only for one member of each class. This improves the algorithm from finding $|\Gamma|^m$ flows to finding $|\Gamma|^{n-1}$ flows (because every equivalence class is uniquely determined by an assignment of forbidden values which is 0 outside of some fixed spanning tree).

A slightly smarter algorithm – used to find $\mathbb{Z}_2^2$-connected graphs which are not $\mathbb{Z}_4$-connected – can be obtained by looking at Observation 2.7 the other way around. It follows that each equivalence class of $\sim_f$ is exactly the coset $\{h_0 + \varphi : \varphi \text{ a flow}\}$ for any h_0 in the class. Therefore if an equivalence class $[x]_{\sim_f}$ is satisfied then for every flow φ there is $h \in [x]_{\sim_f}$ such that φ satisfies h.

Theorem 2.9. *Fix a digraph G and an abelian group Γ. Let $x\colon E \to \Gamma$ be a forbidden mapping. The following statements are equivalent:*

1. *Forbidden mapping x is satisfied.*
2. *Every $y \in [x]_{\sim_f}$ is satisfied.*
3. *For every flow φ, there exists $y \in [x]_{\sim_f}$ satisfied by φ.*

Proof. Equivalence of first two follows from Observation 2.7. For item three we fix a flow φ_x satisfying x. Then flow φ satisfies forbidden mapping $x - \varphi_x + \varphi$. And vice versa if φ satisfies y then x is satisfied by $\varphi - y + x$. □

So we can fix a flow – constant-zero flow being the obvious candidate – and for each equivalence class we test whether some of its members are satisfied by it. This increases the number of tests back to $|\Gamma|^m$ but now each test is just a simple comparison instead of an NP-complete problem.

We can also trade some space for time: We keep a table of all equivalence classes, and instead of enumerating members of all equivalence classes, we enumerate all assignments of forbidden values that are satisfied by the given flow. For each of them we determine its equivalence class and mark that class as satisfied. After enumerating them all we just check whether every equivalence class

is satisfied. This decreases the number of enumerated elements to $(|\Gamma|-1)^m$ but consumes additional 2^{n-1} bits of memory.

Because we were testing subdivisions of cubic graphs we would like to optimize cases of once and twice subdivided edges. Without any additional optimization each subdivision of an edge increases the number of edges by one and hence slows down the described method by a factor of $|\Gamma|-1$. But a subdivision creates an edge 2-cut.

Without loss of generality we may assume that edges of a 2-cut – denote them e_1 and e_2 – are oriented in opposite directions. The value of any flow must be the same on both of them. Hence swapping the forbidden values for edges e_1 and e_2 does not change the set of satisfying flows. Moreover, we may assume that the forbidden values for e_1 and e_2 are different because it is more restrictive than the case when they are the same. This reduces the number of cases from $|\Gamma|^2$ to $\binom{|\Gamma|}{2}$ (i.e., from 16 to 6 for groups of order four). Double subdivision is in our case even simpler because we have three forbidden values and again the most restrictive case is when they all are distinct. So such double-subdivided edge has only one possible value (in our case, where $|\Gamma|=4$).

Now we need to plug this observations into above-described algorithm. Observe that the equivalence classes used in the algorithm do not have to be equivalence classes of $\sim_f$ but we can use classes of any equivalence $\sim$ which is a congruence with respect to satisfiability and which is coarser than $\sim_f$. Being congruence with respect to satisfiability means that either all elements of an equivalence class are satisfiable or none of them is. Being coarser than $\sim_f$ ensures that $[x]_{\sim_f} \subseteq [x]_\sim$ and so if class $[x]_\sim$ is satisfiable that for every flow φ there is some $y \in [x]_\sim$ satisfied by φ. Moreover, we can throw away equivalence classes that are satisfied if some other class is satisfied (of course without creating cycles). E.g. if we have a 2-cut with both forbidden values being 1, then this case is implied by the case with value 1 and any other value.

Notation 2.10. *We let $[A \to B]$ denote the set of all functions from A to B. We use $\uplus$ to denote disjoint union.*

We summarize our approach in Algorithm 2.5 and Theorem 2.12. We also need to work with equivalence classes in the algorithm, so we represent the equivalence with throw-away class as a function

$$\mathcal{C}\colon [E \to \Gamma] \to X \uplus \{\texttt{NULL}\}$$

which assigns to each forbidden mapping an object representing its class (in the practical implementation elements of X are just small integers), `NULL` representing the throw-away class.

The function $\mathcal{C}$ we used is obtained from $\sim_f$ by the following modifications: For each 2-cut we remove all classes (i.e., we set the values of their elements to `NULL`) that forbid the same value on both edges of the cut and merge classes which differ only by swapping values on the edges of the cut. For double-subdivided edges we remove all classes that do not forbid three different values on each double-subdivided edge and than merge all classes that differ only by the order of forbidden values on given subdivided edge. We note that the optimization for double-subdivided edges is essentially equivalent to removing a given subdivided edge:

Observation 2.11. *If graph G contains an edge subdivided $|\Gamma|$ times, it cannot be Γ-connected. If it contains an edge e subdivided $|\Gamma|-1$ times, it is Γ-connected if and only if $G-e$ is Γ-connected.*

Input: Graph G, function $\mathcal{C}\colon [E \to \Gamma] \to X \uplus \{\texttt{NULL}\}$
Output: `YES` if G is Γ-connected, and `NO` otherwise

1 Pick a flow φ_0
2 Create array a indexed by elements of X
3 $a[*] \leftarrow \texttt{false}$
4 **for** $\forall h$ satisfied by φ_0 such that $\mathcal{C}(h) \neq \texttt{NULL}$ **do**
5 $a[\mathcal{C}(h)] \leftarrow \texttt{true}$
6 **end for**
7 **for** $\forall x \in X$ **do**
8 **if** $a[x] = \texttt{false}$ **then return** `NO`
9 **end for**
10 **return** `YES`

Algorithm 2.5: Group connectivity testing

Theorem 2.12. *Fix an abelian group Γ, a digraph G, and a function $\mathcal{C}\colon [E \to \Gamma] \to X \uplus \{\texttt{NULL}\}$ such that:*

1. *for all $x \in X$ there exists $h \in [E \to \Gamma]$ such that $x = \mathcal{C}(h)$,*
2. *for all $h\colon E \to \Gamma$ if $\mathcal{C}(h) = \texttt{NULL}$ then there exits $h'\colon E \to \Gamma$ such that if h' is satisfied then h is also satisfied and $\mathcal{C}(h') \neq \texttt{NULL}$,*
3. *for all $h, h'\colon E \to \Gamma$ if $\mathcal{C}(h) = \mathcal{C}(h')$ then either both are satisfied or none of them is, and*
4. *for all $h\colon E \to \Gamma$ and for all Γ-flows φ holds $\mathcal{C}(h) = \mathcal{C}(h+\varphi)$.*

Then Algorithm 2.5 correctly decides whether G is Γ-connected.

Proof. Obviously, Algorithm 2.5 terminates.

First we prove that if the graph is Γ-connected, then the algorithm outputs `YES`. By contradiction, let $x \in X$ be the element that forced the algorithm to output `NO`. Let $P = \mathcal{C}^{-1}(x)$ be a set of preimages of x. It is nonempty due to Assumption 1, so we can fix some $p \in P$. The mapping p is satisfied by some flow φ_p because G is Γ-connected. The mapping $p' = p - \varphi_p + \varphi_0$ is satisfied by flow φ_0 (Observation 2.7). Also $\mathcal{C}(p') = \mathcal{C}(p) = x$ (Assumption 4), so mapping p' was enumerated by the algorithm and set $a[x]$ to `true`. Contradiction.

Now we prove that if the algorithm outputs `YES`, the graph G is Γ-connected. By contradiction, let $p\colon E \to \Gamma$ be a mapping witnessing that G is not Γ-connected. If $\mathcal{C}(p) = \texttt{NULL}$, Assumption 2 gives us p' which is also unsatisfied and $\mathcal{C}(p') \neq \texttt{NULL}$, otherwise we take $p' = p$. Because $\mathcal{C}(p') \neq \texttt{NULL}$, none of the mappings in the set $\mathcal{C}^{-1}(\mathcal{C}(p'))$ is satisfied (Assumption 3). Hence $a[C(p')]$ was never set to `true`, and the algorithm must have returned `NO`. Contradiction. □

Table 2.6: Time required to test cube subdivided on 2 edges (all 9 possibilities)

Algorithm	Time [s]
Simple (in Python)	48.8
Smart (in C++)	3.65
Smart with subdivision optimization	0.229

Measured on Intel i5 5257U.

2.3 Implementation Notes

Because large part of our work consisted of creating programs for testing group connectivity, we would like to add some implementation notes. Readers interested only in theoretical results may safely skip this section.

Our first implementation of straightforward algorithm was written by the second author during her master thesis work. It was a C++ implementation which was very specialized for the graphs tested (subdivisions of a cube), and a CSP implementation in Sicstus Prolog to double-check the results. Both of these implementations required preprocessed input which made them less than ideal to work with, and also was not fast enough for searching through larger graphs.

Hence we have written a new implementation based on Algorithm 2.5 in Python version 2 [van Rossum and Drake, 1995] built on Sage libraries [The Sage Developers, 2021] which already contain a lot of tools to work with general graphs.[2] Because Python is an interpreted language and as such is slower, we chose to implement performance critical parts of the code in C++ binding them into Python using Cython [Behnel et al., 2011].[3]

At the end of the previous section we have described the function $\mathcal{C}$ that we are using, but we did not specify how to calculate it. The main idea is to fix a spanning tree and transform any forbidden mapping to an equivalent one which is zero outside this tree. To do so we keep a precalculated list of elementary flows. We also need to take care of merged classes created by (doubly-)subdivided edges. For doubly subdivided edges we always assign them the only interesting forbidden values (and remove them from generation of forbidden mappings). For single subdivisions we keep a list of six interesting assignments and assign subdivided edges only values from this list. The effect of these optimizations is shown in Table 2.6.

To double-check our results we also implemented the straightforward algorithm in pure Python. It is called Simple algorithm in Table 2.6. It does just check the definition – for every forbidden assignment (fixed outside of a spanning tree) it finds a satisfying flow (from precomputed list of flows). A repository with both implementations may be found at our department's GitLab

https://gitlab.kam.mff.cuni.cz/radek/group-connectivity-pub.

[2] We used version 2 of Python because Sage was not yet ported to Python 3 at that time.

[3] Do not mistake with CPython – CPython is the reference implementation of Python interpreter, whereas Cython is an optimizing compiler of Python which compiles Python into C (or C++) and then into the native code using a standard compiler like gcc.

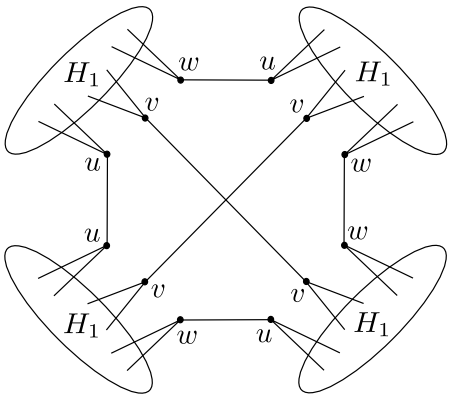

Figure 2.7: A cubic 3-edge-connected graph that is $\mathbb{Z}_2^2$- but not $\mathbb{Z}_4$-connected (for graph H_1 see Figure 2.2)

2.4 Conclusions and Open Problems

Cubic Graphs We have found graphs that show that $\mathbb{Z}_2^2$- and $\mathbb{Z}_4$-connectivity are independent notions. All of the graphs that we have found to certify this do have vertices of degree 2, and Li [2018] can make them 3-edge-connected but only using vertices of degree 4. Therefore, it is natural to ask, whether such graphs exist that are cubic and 3-edge-connected.[4] Moreover Robinson and Wormald [1992] proved that asymptotically almost every cubic graph is Hamitonian so it also has a nowhere-zero $\mathbb{Z}_4$- and $\mathbb{Z}_2^2$-flow which motivates the following question: Is asymptotically almost every cubic graph $\mathbb{Z}_4$- and $\mathbb{Z}_2^2$-connected?

Avoiding Computers Another challenging task is to find a proof that does not use computers. The main obstacle is to find efficient techniques to show that a particular graph is Γ-connected. To prove the converse is much easier: we guess forbidden values $h\colon E \to \Gamma$ and then show non-existence of a flow (see Theorem 2.5).

Complexity Our final question is the complexity of testing group connectivity. The algorithm we have developed is fast enough for our purposes; the required time is exponential, however. To test for group connectivity seems harder than to test for existence of a nowhere-zero flow, which suggests the problem is NP-hard. In fact, we believe it is Π_2^p-complete.

Circumstantial evidence which suggests Π_2^p-completeness of the group connectivity testing are the somewhat dual notions of choosability and group list-colorings. Both of these problems are known to be Π_2^p-complete – proved by Erdős et al. [1980] for choosability, and by Kráľ and Nejedlý [2004] and Kráľ [2005] for group list-colorings. Of those two, group list-colorings are a closer match to the dual of group connectivity, but the graphs used in Kráľ's proofs are non-planar, and we found no way to work around it. So for testing group connectivity we do not know any hardness results.

[4] After the publication of the paper, this question was resolved as described in the next section. The other questions remain open.

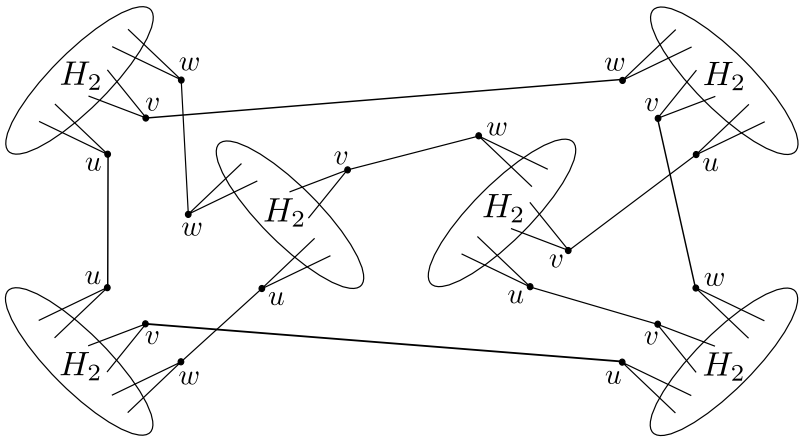

Figure 2.8: A cubic 3-edge-connected graph that is $\mathbb{Z}_4$- but not $\mathbb{Z}_2^2$-connected (for graph H_2 see Figure 2.2)

2.5 Recent Development

Han et al. [2020] used our results (namely graphs H_1 and H_2 shown in Figure 2.2) to construct cubic 3-edge-connected graphs which are $\mathbb{Z}_4$- and not $\mathbb{Z}_2^2$-connected and vice versa. For the sake of completeness we include their results here.

Theorem 2.13 (Han et al. [2020])**.**

- *The cubic 3-edge-connected graph in Figure 2.7 is $\mathbb{Z}_2^2$- but not $\mathbb{Z}_4$ connected.*
- *The cubic 3-edge-connected graph in Figure 2.8 is $\mathbb{Z}_4$- but not $\mathbb{Z}_2^2$ connected.*

Combining this result with the result of Jaeger et al. [1992], the question of Lai et al. [2011] is solved.

Theorem 2.14 (Jaeger et al. [1992])**.** *Let Γ be an abelian group. Then*

- *every 3-edge-connected graph is Γ-connected if $|\Gamma| \geq 6$, and*
- *every 4-edge-connected graph is Γ-connected if $|\Gamma| \geq 4$.*

Problem 2.15 (Lai et al. [2011])**.** *Let $\mathcal{F}(\Gamma)$ be the family of all 3-edge-connected Γ-connected graphs. Is it true that for two abelian groups Γ_1 and Γ_2 if $|\Gamma_1| = |\Gamma_2|$ then $\mathcal{F}(\Gamma_1) = \mathcal{F}(\Gamma_2)$?*

Corollary 2.16 (Han et al. [2020])**.** *Let Γ_1 and Γ_2 be abelian groups such that $|\Gamma_1| = |\Gamma_2|$. Then every 3-edge-connected Γ_1-connected graph is Γ_2-connected if and only if $\{\Gamma_1, \Gamma_2\} \neq \{\mathbb{Z}_4, \mathbb{Z}_2^2\}$.*

We conclude this chapter with another question. The case of 3-edge-connected graphs is fully solved now but it is not obvious what about 2-edge-connected graphs. Is it true that every two abelian groups can be distinguished by some graph? We are not sure about the answer but we are slightly inclined to the positive one:

Conjecture 2.17. *Let Γ_1 and Γ_2 be abelian groups. Then there exists a graph which is Γ_1-connected but not Γ_2-connected.*

3. Graph Homomorphisms and Cayley Graphs

This chapter is based on work which was first presented at Eurocomb 2017 [Hušek and Šámal, 2017] and later extended and published as journal paper Hušek and Šámal [2020].

Definition 3.1. *Let $B \subseteq M$ be symmetric, i.e., $B = -B$. We say that M-flow φ is an (M, B)-*flow *if $\varphi(e) \in B$ for all $e \in E$.*

We require B to be symmetric so that reversing an edge e and changing the sign of $\varphi(e)$ preserves the property of being an (M, B)-flow. Thus, the existence of such a flow depends only on the underlying undirected graph.

The main example is $B = M \setminus \{0\}$; in this case we call an (M, B)-flow a *nowhere-zero M-flow*. The study of nowhere-zero flows was started by Tutte [1954, 1949], main motivation was the fact, that a planar graph has a proper face k-coloring if and only if it has a nowhere-zero $\mathbb{Z}_k$-flow. For a more thorough introduction to the theory that came out of this we refer the reader to Diestel [2017, Chapter 6] or Zhang [1997]. Here we just present the results and notions crucial for our exposition.

Theorem 3.2 (Tutte [1949])**.** *Let $k \geq 2$ be an integer and let f be a $\mathbb{Z}_k$-flow on a graph G. Then there is a $\mathbb{Z}$-flow g on the same graph such that for every edge e we have $f(e) \equiv g(e) \pmod{k}$ and $|g(e)| < k$. Conversely, for each $\mathbb{Z}$-flow g, the flow $f := g \bmod k$ is a $\mathbb{Z}_k$-flow.*

An *M-tension* τ on a digraph G is again a mapping $E \to M$ but the condition is that the oriented sum along every cycle C is zero, explicitly

$$\sum_{e \in C^+} \tau(e) - \sum_{e \in C^-} \tau(e) = 0$$

where C^+ are edges of C with one orientation along the cycle and C^- the edges with the opposite orientation. We define *(M, B)-tension* to be an M-tension which uses only values from a symmetric set $B \subseteq M$.

Before stating our next observation, we need to define the notion of Cayley graph. Given an abelian group M and its symmetric subset $B \subseteq M$ we let $\mathrm{Cay}(M, B)$ denote the graph with vertex set M and with edges $\{uv : u, v \in M, v - u \in B\}$. The notion of a tension defined in the previous paragraph can be equivalently described by its relation to vertex colorings. Consider a mapping $p\colon V \to M$ (usually called a group coloring or in this context a potential). If we define $\tau(uv)$ as $p(v) - p(u)$ for every edge uv (we write $\tau = \delta p$), then τ is a tension. On the other hand, it is easy to show that every tension can be written as δp for some potential p. If τ is an (M, B)-tension, then p only uses values in $B \subseteq M$, thus p is a homomorphism into $\mathrm{Cay}(M, B)$.

For planar graphs, flows and tensions are dual notions – every flow in the primal graph corresponds to a tension in its dual and vice versa. Because a composition of homomorphisms is a homomorphism, the following statement holds:

Observation 3.3. *Let M, M' be abelian groups and $B \subseteq M$, $B' \subseteq M'$ their symmetric subsets. If there is a graph homomorphism from $\mathrm{Cay}(M, B)$ into $\mathrm{Cay}(M', B')$, then every graph with an (M, B)-tension has an (M', B')-tension.*

Many fruitful questions about flows on graphs were motivated by mimicking the properties of coloring in the dual setting Tutte [1954]. In the same spirit, we ask for the dual version of Observation 3.3:

Conjecture 3.4 (DeVos [2007])**.** *Let M, M' be abelian groups and $B \subseteq M$, $B' \subseteq M'$ their symmetric subsets. If there is a graph homomorphism from $\mathrm{Cay}(M, B)$ into $\mathrm{Cay}(M', B')$, then every graph with an (M, B)-flow has an (M', B')-flow.*

This is still an open problem but it holds in some special cases. We start with a few immediate observations that appear in DeVos [2007]. The Conjecture 3.4 holds

- if G is planar (because of duality and Observation 3.3) or
- if $0 \in B'$ (every graph has an $(M', \{0\})$-flow) or
- if $B = M \setminus \{0\}$ and $B' = M' \setminus \{0\}$: Here an (M, B)-flow is just a nowhere-zero M-flow. It is known that the existence of a nowhere-zero flow is monotone in the size of the group [Tutte, 1954].

A generalization of the last example is based on the monotonicity of circular flows. A *circular k/d-flow* is a $\mathbb{Z}$-flow φ such that $d \leq |\varphi(e)| \leq k - d$ for every edge e. It was proved in Goddyn et al. [1998] that every graph with a circular k/d-flow has a circular k'/d'-flow (assuming $k/d \leq k'/d'$).

Let $M = \mathbb{Z}_k$, $B = \pm\{d, d+1, \ldots, k-d\}$, $M' = \mathbb{Z}_{k'}$, $B' = \pm\{d', d'+1, \ldots, k'-d'\}$. Note that by Theorem 3.2 we may equivalently define circular k/d-flow to be a $\mathbb{Z}_k$-flow with values in $\{d, d+1, \ldots, k-d\}$, that is an (M, B)-flow. Thus the result of Goddyn et al. implies that every graph with (M, B)-flow has an (M', B')-flow if and only if $k/d \leq k'/d'$.

The Cayley graph $\mathrm{Cay}(M, B) \cong K_{k/d}$ is frequently denoted as the *circular clique* (also as circular complete graph); similarly, $\mathrm{Cay}(M', B') \cong K_{k'/d'}$. This graph is important in the study of circular coloring, the dual concept of circular flows. It is known that there is a homomorphism from $K_{k/d}$ to $K_{k'/d'}$ if and only if $k/d \leq k'/d'$ (see, for example, Zhu [2001] or its references). Thus, both sides of the conjectured implication are in this setting equivalent to $k/d \leq k'/d'$, hence the conjecture holds for these combinations of groups and their subsets.

We thank the anonymous referee who kindly suggested that our example using circular $(2k+1)/k$-flows extends to any two circular flows.

3.1 New Framework

The structure of homomorphisms from $\mathrm{Cay}(M, B)$ to $\mathrm{Cay}(M', B')$ is hard to describe. Instead we take any mapping $m\colon M \to M'$ (not necessarily a group homomorphism) and let B' be determined by m (so B' is the minimal set for which m is a graph homomorphism). This is achieved by the following technical definition:

Definition 3.5. *Let M, M' be abelian groups and $m\colon M \to M'$ any mapping. For $x \in M$ we define its* homomorphic image

$$\mathcal{H}_m(x) := \{m(a+x) - m(a) : a \in M\}.$$

We omit the index m whenever possible. Observe that in the case of tensions $\mathcal{H}(x)$ is exactly the set of possible images of value x on some edge after composing original tension represented by a group coloring with m:

Observation 3.6. *Let $p\colon V \to M$ be a group coloring and let $m\colon M \to M'$ be any mapping between abelian groups M and M'. Define $p' = m \circ p$, $\tau = \delta p$, and $\tau' = \delta p'$. Then*

- *τ is a M-tension,*
- *τ' is a M'-tension, and*
- *$\forall e \in E : \tau'(e) \in \mathcal{H}(\tau(e))$.*

Property 3.7 (Homomorphism property)**.** *Let G be a (directed) graph and let $m\colon M \to M'$ be an arbitrary mapping between abelian groups. We say that G has* homomorphism property *(HP) for m if for every (M, B)-flow there exists an $(M', \bigcup_{x\in B} \mathcal{H}(x))$-flow. We say that G has HP for group M if it has HP for all mappings m with domain M, and that G has HP if it has HP for all abelian groups.*

The name of the property is due to the fact that m is a graph homomorphism from $\mathrm{Cay}(M, \bigcup_{e\in E} \varphi(e))$ to $\mathrm{Cay}(M', \bigcup_{e\in E} \mathcal{H}(\varphi(e)))$.

Conjecture 3.8 (Reformulation of Conjecture 3.4)**.** *All graphs have homomorphism property.*

Proof of equivalence of Conjectures 3.4 and 3.8. For any mapping $m : M \to M'$, we put $B'' = \bigcup_{x\in B} \mathcal{H}(x)$. As m is a homomorphism of $\mathrm{Cay}(M, B)$ to $\mathrm{Cay}(M', B'')$, Conjecture 3.4 implies Conjecture 3.8. On the other hand, if m is any homomorphism of $\mathrm{Cay}(M, B)$ to $\mathrm{Cay}(M', B')$ (note the different target graph), then $B'' \subseteq B'$. Consequently, every (M', B'')-flow is also an (M', B')-flow and thus Conjecture 3.8 implies Conjecture 3.4. □

The traditional approach to solving flow-related conjectures is to study properties of a hypothetical minimal counterexample. Usually the problem is reduced to cubic graphs by splitting / decontracting vertices. This, however, is not possible with Conjecture 3.4 because decontracting a vertex may create an edge with a new value found nowhere else, modifying B. To overcome this we formulated the following property which is a strengthening of the homomorphism property:

Property 3.9 (Strong homomorphism property)**.** *Let G be a (directed) graph and $m\colon M \to M'$ an arbitrary mapping between abelian groups. We say that G has* strong homomorphism property *(SHP) for m if for every M-flow φ there exists an M'-flow φ' such that $\varphi'(e) \in \mathcal{H}(\varphi(e))$ for all edges e. We say that G has SHP for group M if it has SHP for all mappings m with domain M, and that G has SHP if it has SHP for all abelian groups.*

Note that SHP allows the flow to be zero on some edges but such edges are not interesting because $\mathcal{H}(0)$ is always $\{0\}$. The SHP allows us to study only cubic graphs – we can make any graph (sub)cubic by decontracting its vertices of high degree and if SHP holds for such decontracted graphs then it holds for the original graph too. To state this in a formal way, we need the following technical definition:

Definition 3.10. *We say that a digraph H is a* cubification *of digraph G if H can be obtained from G using following operations:*

1. *decontraction of vertex of degree at least 4 (such that both new vertices have degree at least 3),*
2. *suppression of a vertex of degree 2,*
3. *deletion of a bridge,*
4. *deletion of a loop, and*
5. *deletion of an isolated vertex.*

With this definition we want to show that every non-cubic graph can be reduced to a smaller cubic one. To get this we need to use a slightly non-standard definition of the size of the graph which considers graphs with larger degrees bigger. Suitable definition for us is

$$\Phi := \sum_{v \in V} 3^{\deg v}.$$

Observation 3.11 (Reducibility of SHP to cubic graphs)**.** *Let $m\colon M \to M'$ an arbitrary mapping between abelian groups and let G be a digraph. If some cubification of G has SHP for m, then also G has SHP for m. Moreover for every flow $\varphi : E \to M$ there exists a non-strictly smaller (possibly empty) cubic graph $G' = (V', E')$ and a nowhere-zero flow $\varphi' : E' \to M$ such that if SHP does not hold for φ on G then SHP also does not hold for φ' on G'.*

Proof. To prove the first part, we only need to show that inverse of each operation used in Definition 3.10 does not break SHP:

1. Suppose $G = G_1/e$ and G_1 has SHP for m. Let φ be an M-flow on G. There is a unique extension of φ to G_1, we use φ for this extension as well. (Note that the value of $\varphi(e)$ may be 0.) As G_1 has SHP for M, there is an M'-flow φ' on G_1 such that $\varphi'(e) \in \mathcal{H}_m(\varphi(e))$. The restriction of φ' to $G = G_1/e$ is the desired M'-flow on G.
2. Subdivision of an edge is obvious when the new vertex of degree 2 has both in-degree and out-degree 1. In the other case SHP still holds because $\mathcal{H}(-x) = -\mathcal{H}(x)$.
3. Addition of a bridge does not break SHP because flow on a bridge is always 0 and $\mathcal{H}(0) = \{0\}$.
4. Addition of a loop is also simple because $\mathcal{H}(x)$ is always non-empty and we can assign any value on a loop without affecting the rest of the flow.

5. Addition of an isolated vertex does not change the flow at all.

The moreover part: Note that $\Phi = \sum_{v \in V} 3^{\deg v}$ for every cubification is strictly smaller than Φ of the original graph. To obtain G' we set $G' = G$, $\varphi' = \varphi$, and apply the following operations as long as possible:

1. Remove an edge $e' \in E'$ such that $\varphi'(e') = 0$.

2. Apply some cubification operation on G'.

Because each of the operations decreases $\Phi(G')$, the process terminates. If the resulting φ' was not nowhere-zero, we still could remove an edge with 0 flow, and if G' was not cubic, we could get a non-trivial cubification. □

The SHP is a natural strengthening of HP – we just fix a particular (M, B)-flow φ and try to find an (M', B')-flow φ' with an extra requirement $\varphi'(e) \in \mathcal{H}(\varphi(e))$. Observation 3.6 shows that a variation of SHP for tensions holds in general, so also all planar graphs have SHP due to duality.

With computer aid we found out that not all graphs have a SHP. The smallest graphs without SHP that we found are K_5 and $K_{3,3}$ with a particular $\mathbb{Z}_5$-flow and the universal mapping; see Figure 3.1 and Definition 3.14 below. Due to the universal mapping concept, and given the problematic $\mathbb{Z}_5$-flow, it is actually easy to see by hand that $K_{3,3}$ does not have SHP.

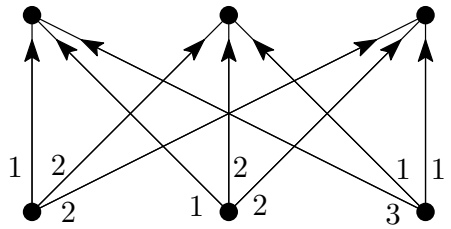

Figure 3.1: A graph with a $\mathbb{Z}_5$-flow for which SHP does not hold

Although SHP does not hold for K_5 and $K_{3,3}$ in general it still holds for groups $\mathbb{Z}_3$ and $\mathbb{Z}_4$ because SHP always holds for groups of size at most 4 (Theorem 3.17). We also tested that SHP holds for Petersen graph and $\mathbb{Z}_5$ (we did not try larger snarks due to computational complexity). This motivates our next conjecture:

Conjecture 3.12 (SHP for minimal groups)**.** *For every graph G the strong homomorphism property holds for group $\mathbb{Z}_k$ where k is minimal such that G admits a nowhere-zero $\mathbb{Z}_k$-flow.*

It is easy to observe that SHP holds for m which are (induced by) a group homomorphism but a more general statement is true:

Observation 3.13. *Let G be a graph and let $m\colon M \to M'$ be some mapping of abelian groups. Let $h\colon M' \to M''$ be a group homomorphism. If SHP (resp. HP) holds for G and m then it also holds for G and $h \circ m$.*

Proof. Let φ' be an M'-flow guaranteed by SHP. For a group homomorphism h it holds $\mathcal{H}_h(x) = \{h(x)\}$. Then $\varphi'' = h \circ \varphi'$ is also an M''-flow and its values satisfy

$$\begin{aligned}\varphi''(e) \in \mathcal{H}_h[\mathcal{H}_m(\varphi(e))] = \{h(m(a + \varphi(e)) - m(a)) : a \in M\} = \\ = \{h(m(a + \varphi(e))) - h(m(a)) : a \in M\} = \mathcal{H}_{hom}(\varphi(e)).\end{aligned}$$ □

3.2 Universal Objects

Observation 3.13 leads us to the definition of a universal mapping such that if HP or SHP holds for this mapping, it also holds for every other mapping.

Definition 3.14 (Universal mapping)**.** *Let M be an abelian group. We define its* universal group $\mathcal{G}_M = \mathbb{Z}^M$ *and its* universal mapping $\mathcal{M}_M \colon M \to \mathcal{G}_M$*:*

$$x \longmapsto g_x$$

where g_x is a vector with 1 *on position x and* 0 *elsewhere.*

The group $\mathbb{Z}^{M\setminus\{0\}}$ (with 0 mapped to 0 instead of g_0) would be sufficient but we choose the definition with $\mathbb{Z}^M$ to simplify the proofs. Note that with this definition $\mathcal{G}_M$ is just a free group generated by elements of M.

Observation 3.15. *The universal mapping is universal for both HP and SHP, i. e., if for a given graph (and flow) HP (resp. SHP) holds for the universal mapping $\mathcal{M}_M$ then it holds for M.*

Proof. Let $m \colon M \to M'$ be any mapping. We can also interpret m as a homomorphism $m_{\text{ext}} \colon \mathcal{G}_M \to M'$ – mapping m defines values of generators e_x and hence it can be uniquely extended into mapping on the whole group which is a homomorphism (here we are using the fact that $kg_x = 0 \Rightarrow k = 0$ for $x \neq 0$). Moreover $m = m_{\text{ext}} \circ \mathcal{M}_M$ so Observation 3.13 finishes the proof. □

There also exists a universal object on the left-hand side – a universal flow $\mathcal{F}$ (which is just the flow into a free group generated by edges outside of some fixed spanning tree) – but $\mathcal{G}_\mathcal{F}$ has infinitely many generators so we have not found any reasonable way to work with it. Also note that although the universal group is infinite, (S)HP holds for the universal group $\mathbb{Z}^M$ if and only if it holds for $\mathbb{Z}_k^M$ for any $k > \Delta(G)$.

3.3 Partial Results

In this section we prove SHP for some special cases of the mapping or the group.

Theorem 3.16 (Mappings with one "hole")**.** *Strong homomorphism property holds for mappings $m \colon \mathbb{Z}_k \to \mathbb{Z}_l$ defined by $m(x) = ax \bmod l$ where $a \in \mathbb{Z}$. (Here we interpret elements of $\mathbb{Z}_k$ as integers $0, 1, 2, \ldots, k-1$.)*

Proof. Note that mapping m is a composition of mappings $m_1 \colon \mathbb{Z}_k \to \mathbb{Z}$ defined by $m_1(x) = x$ and $m_2 \colon \mathbb{Z} \to \mathbb{Z}_l$ defined by $m_2(x) = ax \bmod l$, and that m_2 is a group homomorphism. Hence we only need to show that SHP holds for m_1, the rest follows from Observation 3.13.

So we need to show that for every $\mathbb{Z}_k$-flow φ there exists a $\mathbb{Z}$-flow φ' such that $\varphi'(e) \in \mathcal{H}_{m_1}(\varphi(e)) = \{\varphi(e), \varphi(e) - k\}$. This, however, is a well-known result of Tutte [1949]. □

Theorem 3.17. *Strong homomorphism property holds for groups $\mathbb{Z}_2$, $\mathbb{Z}_3$, $\mathbb{Z}_2^2$, and $\mathbb{Z}_4$.*

Proof. Due to Observation 3.11 we know that minimal counter-example is a cubic graph G and nowhere-zero flow φ. We denote the generators of the right-hand side free group a, b, c, ...

- $\mathbb{Z}_2$: The only graph cubic with nowhere-zero $\mathbb{Z}_2$-flow is the empty graph, for which the claim holds.

- $\mathbb{Z}_3$: Let the mapping m be $0 \mapsto a$, $1 \mapsto b$, and $2 \mapsto c$. So $\mathcal{H}(1) = \{b-a, c-b, a-c\}$. A cubic graph has a nowhere-zero $\mathbb{Z}_3$-flow if and only if it is bipartite. (To see this, notice that by changing the orientation of the edges we may assume all flow-values are equal to 1, thus the vertices are either sources or sinks and no two sinks (neither two sources) can be connected by an arc.) So we make all edges directed from one partition to the other and split them into 3 perfect matchings. Observe that either $\varphi \equiv 1$ or $\varphi \equiv 2$ in which case we flip the orientation of edges to get the $\varphi \equiv 1$. We assign one of the following flow values to each matching: $b-a$, $c-b$, $a-c$.

- $\mathbb{Z}_2^2$: Let the mapping m be $00 \mapsto a$, $01 \mapsto b$, $10 \mapsto c$, and $11 \mapsto d$. Then

$$\begin{aligned}\mathcal{H}(01) &= \{\pm(a-b), \pm(c-d)\},\\ \mathcal{H}(10) &= \{\pm(a-c), \pm(b-d)\},\\ \mathcal{H}(11) &= \{\pm(a-d), \pm(b-c)\}.\end{aligned}$$

  Let $C_1, C_2, C_3 \colon E \to \{0, \pm 1\}$ be integer flows on G satisfying the following relations.

$$\begin{aligned}C_1(e) = 0 &\Leftrightarrow \varphi(e) = 01,\\ C_2(e) = 0 &\Leftrightarrow \varphi(e) = 10,\\ C_3(e) = 0 &\Leftrightarrow \varphi(e) = 11.\end{aligned}$$

  That is, the support of C_1 is the collection of circuits formed by edges e with $\varphi(e) \in \{10, 11\}$; the signs are chosen arbitrary but consistently around each of the circuits. (Similarly for C_2, C_3.) We define $\psi \colon E \to \mathbb{Z}^4$ (recall that $a = (1,0,0,0) \in \mathbb{Z}^4$, and b, c, d are defined similarly):

$$\psi = \frac{C_1+C_2+C_3}{2}a + \frac{C_1-C_2-C_3}{2}b + \frac{-C_1+C_2-C_3}{2}c + \frac{-C_1-C_2+C_3}{2}d.$$

  It is easy to check that ψ is a $\mathbb{Z}^4$-flow and $\psi(e) \in \mathcal{H}(\varphi(e))$.

- $\mathbb{Z}_4$: We observe that every vertex (with all incident edges in the same direction) has either values $2,1,1$ or $2,3,3$. Hence edges with value 2 are a perfect matching. When we remove them we obtain disjoint union of circuits and we modify orientation of remaining edges so they are directed along circuits. With this orientation values around every vertex are $1,2,3$ so both edges with value 1 and edges with value 3 are a perfect matching.

  Let m be $0 \mapsto a$, $1 \mapsto b$, $2 \mapsto c$, and $3 \mapsto d$. Then

$$\mathcal{H}(1) = \{b-a, c-b, d-c, a-d\},$$

$$\mathcal{H}(2) = \{\pm(c-a), \pm(d-b)\},$$
$$\mathcal{H}(3) = \{d-a, a-b, b-c, c-d\}.$$

Let $C_1, C_2, C_3 \colon E \to \{0, \pm 1\}$ be a 3-CDC of G defined:

$$e \in C_1 \Leftrightarrow \varphi(e) \neq 1,$$
$$e \in C_2 \Leftrightarrow \varphi(e) \neq 2,$$
$$e \in C_3 \Leftrightarrow \varphi(e) \neq 3.$$

Observe that we can choose orientation of C_2 such that no edge has value -1 in C_2. And we define $\psi \colon E \to \mathbb{Z}^4$ (recall that $a = (1,0,0,0) \in \mathbb{Z}^4$, and b, c, d are defined similarly):

$$\psi = \frac{-C_1+C_2-C_3}{2}a + \frac{C_1-C_2-C_3}{2}b + \frac{C_1+C_2+C_3}{2}c + \frac{-C_1-C_2+C_3}{2}d.$$

It is easy to check that ψ is a $\mathbb{Z}^4$-flow and $\psi(e) \in \mathcal{H}(\varphi(e))$. □

3.4 Connection to CDC

As we show below, flows obtained from HP or SHP can be easily transformed into an oriented cycle double cover. This is one of the major open questions in the study of cycle spaces in graphs. Moreover if G has SHP for M and a nowhere-zero M-flow then the obtained CDC is orientable and has only $|M|$ cycles. This increases importance of Conjecture 3.8 and of determining for which graphs and groups does SHP hold.

Theorem 3.18 (Universal group and CDC)**.** *Let M be any abelian group. If a graph G has a flow in $\mathcal{G}_M$ using only values $\bigcup_{x \in M \setminus \{0\}} \mathcal{H}_{\mathcal{M}_M}(x)$ then it has an orientable cycle double cover using $|M|$ cycles.*

Proof. Denote $H = \bigcup_{x \in M \setminus \{0\}} \mathcal{H}(x)$. Observe that all elements of H are of form $g_a - g_b$ for some $a, b \in M$ and those a, b are unique. Fix an H-flow φ. We define directed cycles (as mappings $E \to \{-1, 0, 1\}$) $\mathcal{C}_x(e) := (\varphi(e))_x$ and claim that $C = \{\mathcal{C}_x\}_{x \in M}$ is an orientable cycle double cover. From definition C covers each edge twice, once in each direction, and every $\mathcal{C}_x$ is a flow with values $\{-1, 0, 1\}$ because it is a composition of a flow and group homomorphism so it is a cycle. □

Corollary 3.19. *If a graph has HP with respect to $\mathbb{Z}_k$ and nowhere-zero $\mathbb{Z}_k$-flow, then it has an orientable cycle double cover with at most k cycles.*

Seymour [1981] proved that every graph without a bridge admits a nowhere-zero $\mathbb{Z}_6$-flow which combined with the previous corollary for $\mathbb{Z}_6$ gives us the following corollary.

Corollary 3.20. *Conjecture 3.4 (and equivalently Conjecture 3.8) implies that every bridgeless graph has an orientable cycle double cover with at most 6 cycles.*

4. Counting Double Covers

Parts of this and the following two chapters were presented at EuroComb 2021 as Hušek and Šámal [2021]. Several recent results and conjectures study counting versions of classical existence statements. Esperet et al. [2011] proved Lovász–Plummer conjecture:

Theorem 4.1 (Esperet et al. [2011])**.** *Every bridgeless cubic graph has exponentially many perfect matchings.*

A similar result for colorings of planar graphs was proven by Thomassen [2007b]:

Theorem 4.2 (Thomassen [2007b])**.** *Every planar graph has exponentially many (list) 5-vertex-colorings.*

Thomassen [2007a] also conjectured existence of exponentially many 3-vertex-colorings of triangle-free planar graphs. He gave a subexponential bound that was later improved by Asadi et al. [2013]. However, the conjecture stays open. By duality, these results and conjecture can be equivalently stated for the number of nowhere-zero $\mathbb{Z}_5$-flows (or $\mathbb{Z}_3$-flows) of planar (4-edge-connected) graphs. Dvořák et al. [2017] extended this to non-planar graphs:

Theorem 4.3 (Dvořák et al. [2017])**.** *Every 3-edge-connected graph has exponentially many $\mathbb{Z}_3$-flows and exponentially many $\mathbb{Z}_2 \times \mathbb{Z}_2$-flows.*

We ask the same question for cycle double covers of cubic graphs. We show that counting cycle double covers usually allows "cheating" by splitting a cycle consisting of more circuits into many cycles, and for this reason we count circuit double covers instead. We give an exponential bound for planar graphs (Theorem 6.14) and almost exponential bound for graphs on any other fixed surface (Corollary 4.14).

The structure of this and the following two chapters is the following: We first discuss why we chose circuit double covers (or CDC for short) over cycle double covers. Then we show a construction which turns a CDC given by an embedding into a surface into many CDCs. This gives the almost exponential bound but we observe that this technique cannot be applied to all graph sequences – in particular it cannot be applied to Flower snarks.

In the next chapter we formulate a general framework to calculate graphs parameters using linear algebra. Using this framework and a computer we show in Chapter 6 that Flower snarks have exponentially many CDCs and we also prove that planar graphs have exponentially many CDCs. The Chapter 6 also contains a description of the implementation of the framework we used as its correctness is essential for the presented results.

4.1 Circuit vs. Cycle

The existential questions usually ask for a cycle double cover. The purpose of this section is to overview the differences between cycle and circuit double covers (Definition 1.4) and explain why we choose to count circuit double covers. Before that we overview the options:

1. circuit double covers,
2. cycle double covers as (multi)sets of cycles (Definition 1.1),
3. cycle double covers as colorings of circuits or
4. k-cycle double covers for either definition of cycle double cover.

An advantage of defining a cycle double cover to be a coloring of its circuits would probably be simpler calculations as it is easier to count colorings than partitions. But as we noted already in Chapter 1, this definition allows creation of new cycle double covers by just permuting colors. This might be acceptable for a fixed number of colors but leads to the infinite number of cycle double covers when the number of colors is not bounded.

Hence we would be forced to fix the number of colors c. Then the counting is quite straightforward – such cycle double covers can be modeled by $\mathbb{Z}_2^c$ flows with exactly two ones in every used value and those can be counted by, e.g., vertex models (Section 5.5). On the other hand if we tried to apply the framework from Chapter 5 the number of boundaries would be exponential (which is better than for circuit double covers as shown in Observations 6.2 and 6.3) but the base of the exponential is $\binom{c}{2}$ which is too large for practical purposes.

Hence "cycle double covers as coloring of circuits" is not a viable option and we stick with our original definition of cycle double cover. Counting general cycle double covers also seems as a very hard problem. Note that the usual trick – treating a cycle double cover as special flows over $\mathbb{Z}_2^k$ – does not work here for two reasons: Firstly it leads to the abandoned definition of cycle double covers as colorings of circuits and secondly without an upper bound on k we would need to work with $\mathbb{Z}_2^{\mathbb{N}}$. General cycle double covers also cannot be counted by vertex models (Section 5.5) because Observation 4.4 shows that their number can be superexponential even for cubic planar graphs. And a straightforward linear representation (Definition 5.13) of it would not be finite and it would be build on the top of the linear representation for circuit double covers (Section 6.1).

Also cycle double covers still kind of overcount. This can also be considered a feature because it counts "simplier" (composed from shorter circuits) circuit double covers more times so it gives us some extra information. Another option is counting k-cycle double covers for some fixed k. This is obviously simpler problem than general cycle double covers. The problem is setting the right k. As shown below on the example of planar graphs (Observation 4.4) if the graph has a $(k-1)$-cycle double cover then the number of its k-cycle double covers is exponentially larger than the number of its $(k-1)$-cycle double covers. So choosing $k = 5$ would heavily favor edge 3-colorable graphs.

Counting circuit double covers does not suffer from any overcounting due to coloring or grouping things in a different ways. It also does not seem any harder than the previous two options. Hence we choose to count circuit double covers.

Observation 4.4. *The circuit double cover given by an embedding of a graph into the plane can be made into* $2^{\Omega(n_3)}$ *many 5-cycle double covers and* $2^{\Omega(n_3 \log n_3)}$ *cycle double covers just by grouping circuits into cycles in different ways where* n_3 *is number of vertices of degree at least 3.*

Proof. Any graph with a vertex of degree 1 does not have any CDC and subdividing an edge (i.e., addition of vertex of degree 2) does not change number of

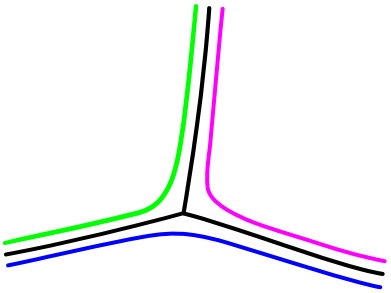

Figure 4.1: CDC around a vertex of degree 3

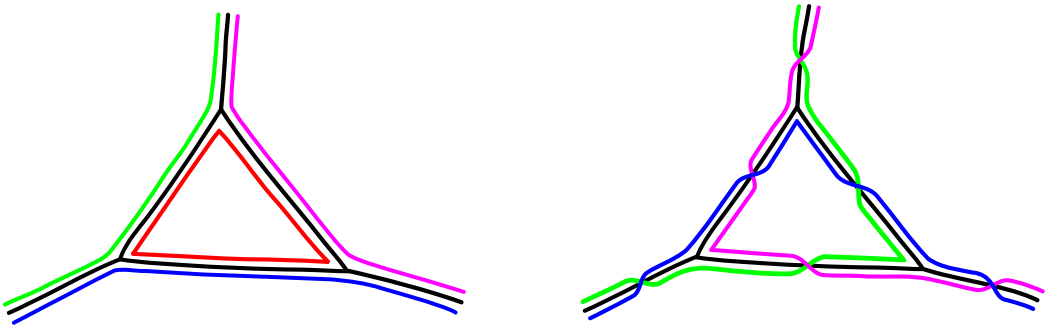

Figure 4.2: Two possible CDCs of a triangle gadget

CDC so we may assume that the graph has minimum degree at least 3. Hence the number of faces is $\Omega(n)$.

Faces of bridgeless planar graph can be 4-colored (due to famous Four color theorem Appel and Haken [1977], Appel et al. [1977]). Some color contains at least 1/4 of the faces which is $\Omega(n)$. For 5-CDC we split this color class into two cycles and for each circuit (except some fixed one to be able to distinguish the cycles) we have a binary choice, hence there is $2^{\Omega(n)}$ such 5-CDCs. For a general CDC we split this color class into any number cycles which leads to the Bell number[1] B_k where $k = \Omega(n)$ is the size of the color class. As a simple lower bound we can take half of the color class to create cycles and divide the rest of the color class among them in an arbitrary fashion leading to $\Omega(n)^{\Omega(n)} = 2^{\Omega(n \log n)}$ CDCs. □

4.2 Representation of Circuit Double Covers

Circuit double covers are usually represented as (multi)sets of circuits. Although this is a natural representation, it has several downsides. The most important is that there is no simple way to do local modifications in this representation. For an alternative representation we use the fact that for cubic graphs any CDC around any vertex looks the same (Figure 4.1). Because all the vertices look the same, the only thing that can change is how are the walks connected to each other at the edges.

There are two ways to join the walks at each edge. To be able to distinguish these two ways we assume that the graph is drawn into a plane, and then the

[1]The Bell number B_k is the number of possible partitions of a set of size k. The name was introduced by Becker and Riordan [1948]. It is known that $B_k = 2^{\Theta(k \log k)}$.

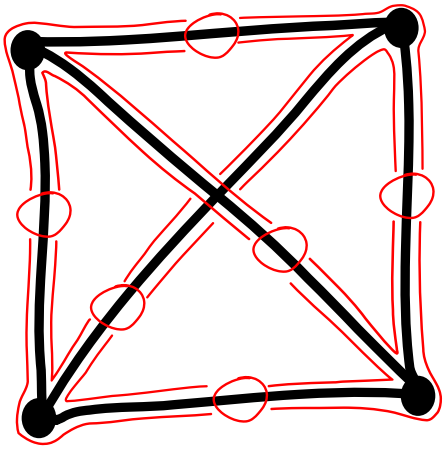

Figure 4.3: A drawing of K_4

two ways correspond to crossing or not crossing the walks at the edge. This is shown in Figure 4.3. We require that all the vertices are distinct points, edges do not pass through the vertices and do not touch but of course they can cross each other. This drawing serves us as a way to fix the rotation system (the order of edges around each vertex). (Alternatively we can view each such a configuration as a *ribbon graph* – see, e.g., Ellis-Monaghan and Moffatt [2013] section 1.1.4.) Every CDC of a cubic graph can be represented this way which leads to the following simple but important observation:

Observation 4.5. *A cubic graph with n vertices has at most $2^{3n/2}$ circuit double covers.*

Proof. A cubic graphs has $3n/2$ edges, there are two ways how to join the walks at every edge and every CDC can be described this way. □

4.3 The Flower Construction

First we present a construction which starts with a graph embedded into some surface and creates almost exponentially many circuit double covers by doing local changes to the CDC given by the embedding. Despite its name, this construction is not related to Flower snarks in any way. The basic idea is to choose a face f and modify the CDC on f and its neighbours (let n_f denote the number of its neighbour faces) so we get $2^{\Omega(n_f)}$ CDC.

This is shown in Figure 4.4 – every small circle there denotes a choice whether to cross the walks or not, leading to exponentially many CDCs (most of the choices lead to 3 circuits but some lead to one or two self-touching walks and we deal with them below). Note that when we redo the CDC on the flower, we cover the outer edges only once as they are once covered from the outside and we do not want to modify that part of the CDC. We model this by taking the flower as a standalone planar graph embedded in such a way that the rest of the original graph would be in the outer face and counting all CDCs which fix the walk corresponding to the outer face. We call such objects *outer-fixed CDCs* (but we will sometimes omit "outer-fixed" as for the flowers we consider only the outer-fixed CDCs).

Definition 4.6 (Flower)**.** *Let G be an embedding of a graph into some surface. We say that a face f of size k together with its neighbour faces are a* k-flower with center f *if the following is satisfied:*

1. $k \geq 3$,
2. *f shares exactly one edge with each of its neighbour faces,*
3. *consecutive neighbour faces also share exactly one edge, and*
4. *non-consecutive neighbour faces do not share any edge.*

To ensure that the result is indeed a CDC with 3 circuits we cannot do arbitrary choices everywhere but the case analyzes in the Figure 4.5 shows that the last 3 choices are enough to ensure a correct CDC. We number the walks on the right side 1, 2, 3. All the choices outside of the Figure 4.5 are already made, so the walks appear on the left side in some order. The upper case in the Figure 4.5 shows what happens if the walk 1 is on top on the left – it will go down so in the middle we must not cross. At the left we still have arbitrary choice but the right choice is forced – there is exactly one correct solution (denoted by full circle in the figure). The bottom case is similar but with walk 1 being on the lower edge.

So given 3 consecutive choice-points we can always make the walks into a CDC. We also still have the original CDC which is important for the flowers of size 3 because we will need to have at least 2 CDCs on every flower. The exact number of outer-fixed CDCs of a flower is determined in Theorem 6.4 but it requires additional definitions so for now we state a lower bound good enough for the construction:

Observation 4.7. *The flower of size k has at least $2^{k-3}+1$ outer-fixed circuit double covers.*

Proof. The "+1" is the original CDC given by the embedding. Other CDCs are consisting of 3 circuits following the idea in the Figure 4.4 which has k binary choice-points. The case analysis in the Figure 4.5 and its description above show that up to 3 choice-points are required to ensure that result is a CDC and hence they cannot be chosen arbitrarily which leads to "-3" in the exponent. □

Before choosing the flowers we need to ensure that every face defines a flower. It is easy to see that if a k-face is not the center of a flower then either the graph

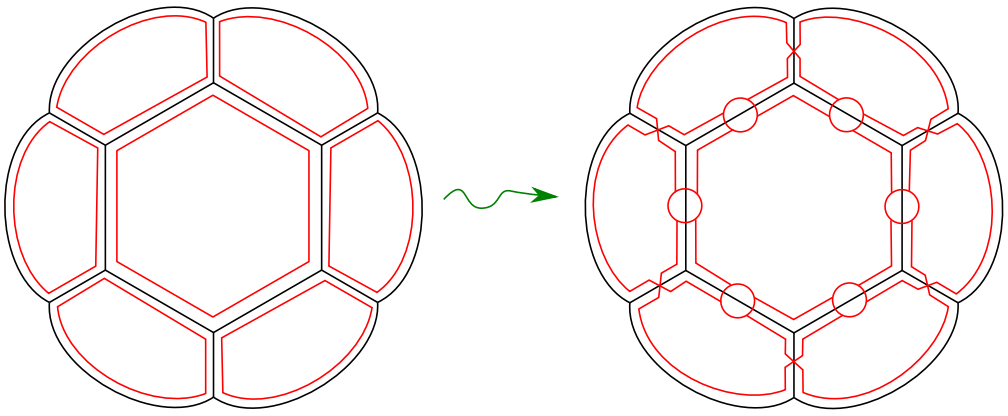

Figure 4.4: The basic idea of the flower construction. Circles denote the possible choices.

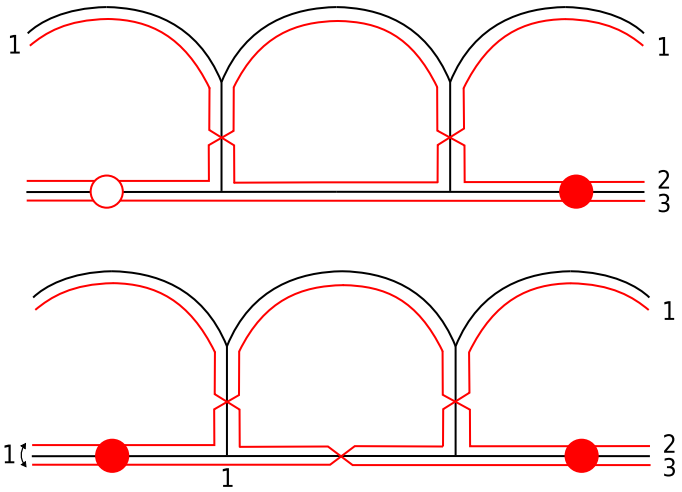

Figure 4.5: Finishing the flower. Full circles denote forced choices.

can be reduced to smaller graph or there must be a closed curve cutting only a few edges:

- If $k = 1$, then the boundary of f is a loop which in a cubic graph implies existence of a bridge and as such it is not interesting for us. If $k = 2$ then f is bounded by two parallel edges which leads to a 2-cut (except the trivial case of three parallel edges on two vertices) and we can apply Observation 4.8 before proceeding further.
- If f shares two or more edges with some face p then there is a closed curve cutting only two of the edges shared by them.
- We also find a similar curve if two consecutive neighbor faces of f share more than one edge.
- If two nonconsecutive neighbor faces of f – denote them p_1 and p_2 – share an edge then there is a closed curve passing through faces f, p_1 and p_2 and cutting only the three edges that the faces share.

In the planar case this means a small cut in the graph – either cut of size 2 or non-trivial cut of size 3. On general surface such a curve does not need to be a cut but it might prove small representativity (Definition 1.67) of the embedding instead. We do not know how to work around the small representativity but we can deal with the small cuts. We denote $\nu(G)$ the number of circuit double covers of the graph G.

Observation 4.8 (2-cut)**.** *Let G be a cubic graph with a cut of size 2 and denote G_1 and G_2 graphs obtained by contracting one side of the cut and suppressing the new vertex of degree 2. Then $\nu(G) = 2\nu(G_1)\nu(G_2)$.*

Proof. There is only one possibility how any CDC looks on a 2-cut. Denote the walks on G_1 ending in the cut-edges w_{11} and w_{12} and the walks on G_2 w_{21} and w_{22}. There are two ways to join them together: either w_{11} with w_{21} and w_{12} with w_{22} or w_{11} with w_{22} and w_{12} with w_{21}. But for these way to really be distict, we need to ensure that we can distinguish both w_{11} from w_{12} and w_{21} from w_{22}.

But we for sure can do that because the cut edges are incident to vertices of degree 3. □

Observation 4.9 (3-cut)**.** *Let G be a graph with cut of size 3 and denote G_1 and G_2 graphs obtained by contracting one side of the cut. Then $\nu(G) = \nu(G_1)\nu(G_2)$.*

Proof. There is only one possibility how any CDC looks on a 3-cut and it also uniquely determines whether walks on each of the cut edges cross or not. Hence we choose some CDC on G_1 and some CDC on G_2 and together they are a CDC on G. □

4.4 Lower Bounds

Now we use the flower construction to obtain almost exponential lower bounds for graphs on surfaces. To choose a suitable set of flowers we need their centers to be at distance 3 or more. This is well modeled by the vertex coloring of the square of the dual graph. For general surfaces we approximate $\chi(G^2) \leq \Delta^2 + 1$ but for the plane there is much better bound:

Theorem 4.10 (Molloy and Salavatipour [2005])**.** *The chromatic number of the square of a planar graph G with maximum degree Δ is $\chi(G^2) \leq \left\lceil \frac{5}{3}\Delta \right\rceil + 78$.*

Theorem 4.11. *Every bridgeless cubic planar graph with n vertices has $2^{\Omega(\sqrt{n})}$ circuit double covers.*

Proof. We proceed by induction by n. If the graph is not cyclically 4-edge-connected we apply Observation 4.8 or 4.9.

If it is, we distinguish two cases depending on Δ, the maximum size of a face: If $\Delta \geq \sqrt{n}$, we choose the largest face to be the center of a flower and this flower itself gives us $2^{\Omega(\sqrt{n})}$ CDCs. Or $\Delta < \sqrt{n}$ and by Theorem 4.10 applied to the dual we can choose flowers in such a way that there is at least $n/O(\sqrt{n}) = \Omega(\sqrt{n})$ of them. Every flower has at least two CDCs so in total we have $2^{\Omega(\sqrt{n})}$ CDCs. □

For general surfaces we have to also consider that Euler characteristic χ influences the number of faces (note that χ is negative except for plane, projective plane, torus and Klein bottle) but otherwise the proof is similar:

Theorem 4.12. *Every bridgeless cubic graph with embedding into a surface of Euler characteristic χ with representativity at least 4 has $2^{\Omega(\sqrt[3]{n+2\chi})}$ circuit double covers.*

Proof. The number of the faces is $f = \frac{1}{2}(n + 2\chi)$ due to Euler's formula. If the graph is not cyclically 4-edge-connected we apply Observation 4.8 or 4.9. A closed curve on the surface which crosses the graph only in the cut edges is a separating curve (otherwise there is a face which is not a disk) so we obtain embeddings of the smaller graphs and we can proceed. For the number of the faces f_1, f_2 of the smaller graphs it holds $f_1 + f_2 = f + 2$ in the case of a 2-cut resp. $f + 3$ for a 3-cut. So we can proceed by induction.

Now lets assume G is 4-edge-connected. We proceed by induction by the number of faces.

Because we ensured that there are no small cuts and excluded the embeddings with small representativity, every face is a center of a flower. Again we distinguish two cases depending on the maximum size of a face Δ: If $\Delta \geq \sqrt[3]{n+2\chi}$, we choose the largest face to be a center of a flower and we are done. Otherwise we color the square of the dual with $\Theta((\sqrt[3]{n+2\chi})^2)$ colors so the largest color class has size $\Omega(\sqrt[3]{n+2\chi})$ and we choose it as centers of the flowers. □

There also exists an analogue of Theorem 4.10 for every fixed surface so for a fixed surface we can improve the exponent:

Theorem 4.13 (Amini et al. [2013])**.** *Let σ be a fixed surface. Then there exists $c \in \mathbb{R}$ such that the chromatic number of the square of a graph G with embedding into σ with maximum degree Δ is $\chi(G^2) \leq c\Delta$.*

Corollary 4.14. *Let σ be a fixed surface. Every bridgeless cubic graph with embedding into the surface σ and with representativity at least 4 has $2^{\Omega(\sqrt{n})}$ circuit double covers.*

This theorem has two potential weaknesses: It might happen that the Euler characteristic of the embeddings will decrease too fast leading to a constant number of faces. We are currently not aware of any graph sequence for which this happens for all the possible embeddings.

The second weakness is that all the embeddings might have small representativity. Mohar and Vodopivec [2006] proved that this is the case for Flower snarks (Definition 1.11):

Theorem 4.15 (Mohar and Vodopivec [2006])**.** *All embeddings of a Flower snark J_k for $k \geq 5$ have representativity at most 2.*

It is therefore natural to ask how many CDCs do Flower snarks have. We answer this question in the following sections. Also the theorems give better bounds if the graph either has a face of a linear size or has the sizes of the faces bounded by a constant. This leads to a question whether we do not loose too much by just changing the strategy in the middle and whether there is some better way.

We answer this question negatively at least for the planar case. The counterexample is an antiflower. We construct an antiflower of size k the following way:

- Take a flower of size k,
- add another layer of faces (we call them green faces) around the flower so every non-central face (the purple faces) of the flower touches k new faces,
- contract the outer face into a single vertex, and
- expand this vertex into a path so the graph is cubic and each of the green faces touches at most 4 other green faces.

Figure 4.6 shows an antiflower of size 4, the outer face in the figure is the central face of the flower used in the construction. Generally an antiflower of size k has k purple faces and $k^2 - k$ green faces so each purple face is incident with k green faces.

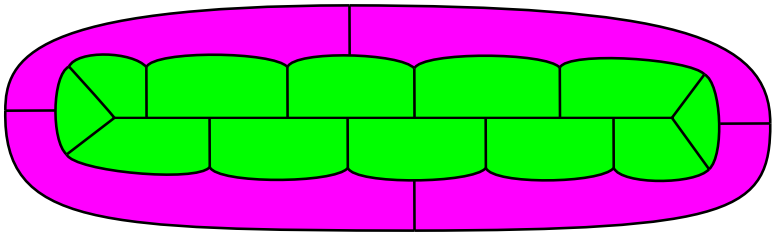

Figure 4.6: Antiflower of size four

Observation 4.16. *Using the flower construction, no matter how we choose centers of the flowers, we obtain at most $2^{O(\sqrt{n})}$ CDCs for the antiflower with n vertices (the size of the antiflower is $\Theta(\sqrt{n})$).*

Proof. Note that the antiflowers are 3-edge-connected so the presented embedding into the plane is the only possible (modulo changing which face is the outer one, proved by Whitney [1932]). We analyse the possible choices of flowers. If we choose outer face as the center of the flower then there is no other flower disjoint with it and it has size $\sqrt{n}$. If we choose a purple face then the outer face is part of the flower so we can choose at most one purple face and the flower has size $\sqrt{n}$ again.

Only the green faces remain. But every flower with center in the green face has size at most 6 as it neighbours with at most 4 other green faces and at most 2 purple faces. And when we select a green face, we cannot select another green face which neighbours with the same purple face. So we can select at most $\sqrt{n}$ green faces at once. □

We did not prove any lower bound specially for antiflowers but they are planar graphs so the general exponential bound proved below (Theorem 6.14) applies.

5. Representations

We want to study a way to effectively calculate number of CDCs of graphs belonging to a graph sequence. The original motivation is examination of the flower snarks for which Theorem 4.12 does not apply but this study also leads to other interesting results like exponential lower-bound on number of CDCs of planar graphs. In this chapter we prepare the general framework for this.

The presented framework is more detailed than it is necessary to formulate the mentioned results. We do this to get a notion of the best possible representation of some graph parameter (Theorem 5.18). This representation is unique up to isomorphism (Observation 5.23). We can also verify that a given representation is the best one if we allow free edges (Corollary 5.28).

We formulate our definitions in the framework of universal algebras. The basic idea is to construct graphs by joining gadgets, and to extend the definition of a graph parameter so it can be effectively computed along this construction.

5.1 Gadgets and Gadget Algebra

It is important to note that all vertices and edges have identity – i.e., given a graph with two vertices and three parallel edges, we can tell which edge is which.

A *gadget* is a graph with some half-edges – a half-edge is an edge which is incident to only one vertex in the gadget and its other end is "floating" so it can be connected to other half-edge. This notion is similar to "graphs with outgoing edges" of Szegedy [2007], k-fragments of Schrijver [2015], networks of Kochol [2006] and probably many others.[1]

We define the *size of the gadget* to be the number of half-edges it has and denote it $|g|$ for a gadget g. We also allow so called free edges – a *free edge* is an edge whose both ends are half-edges (such an edge contributes 2 to the size of the gadget). The half-edges of the gadget are always labeled by numbers $1, \ldots, s$ (each half-edge has a different number). We say k-gadget instead of gadget of size k.

Definition 5.1 (Gadget)**.** *A* k-gadget *is a graph with* k *half-edges which are labeled with numbers 1 to* k *(each used once). Size of the gadget is the number of its half-edges. Two gadgets are isomorphic (denoted* $\cong$*) if there exists an isomorphism of the underlying graphs which maps each half-edge to a half-edge with the same label.*

At this point we could define the class of all gadgets, of all k-gadgets etc. But we often need to restrict ourselves to a subclass of graphs like cubic graphs. Hence we use $\mathcal{G}^*$ for any class of gadgets which is closed on all elementary joins (defined below) and define $\mathcal{G}^k$ to be all the k-gadgets in class $\mathcal{G}^*$. Note that graphs $\mathcal{G}$ are exactly $\mathcal{G}^0$, the gadgets of size zero. We create gadgets from other gadgets by the following three types of elementary operations defined for all $k, k' \in \mathbb{N}$, we call them *elementary joins*:

[1] It seems – similarly to Ackerman's function – that every author defines their own version of it, very similar yet not the same, and we feel obliged to uphold this tradition.

1. *Disjoint union* $\uplus^{k,k'} : \mathcal{G}^k \times \mathcal{G}^{k'} \to \mathcal{G}^{k+k'}$ which result is a gadget whose vertices, edges and half-edges are disjoint unions of those of the input gadgets, only the labels of the half-edges of the second gadget are shifted by the size of the first one. The size of the result is the sum of the sizes of the operands.

2. *Join of two half-edges*, denoted $\mathcal{J}^k_{i,j} : \mathcal{G}^k \to \mathcal{G}^{k-2}$ for all $k \geq 2$ joining half-edges i and j which replaces the two half-edges by an edge joining their endpoints. This decreases the size by two and shifts the labels so there are no gaps.

3. *Permutation of half-edge labels*, denoted $\pi^k[\,.\,] : \mathcal{G}^k \to \mathcal{G}^k$. For example $\pi^3[2,3,1](g)$ means replacing label 1 with 2, label 2 with 3 and label 3 with 1 and requires g to be a gadget of size three.

We usually omit the k and k' as they can be inferred from context when needed.

The usual way to describe $\mathcal{G}^*$ is to choose a set of gadgets and take the smallest $\mathcal{G}^*$ which contains them all. If we do not specify $\mathcal{G}^*$ we work with, then the claims hold for any choice of $\mathcal{G}^*$.

The base gadgets we usually use are a k-vertex $\mathcal{V}_k$ ($|\mathcal{V}_k| = k$, a vertex with k half-edges) and a free edge $\mathcal{F}$ ($|\mathcal{F}| = 2$). For technical reasons we allow an empty gadget and also $\mathcal{J}_{1,2}(\mathcal{F})$ which is a vertexless loop. For example we can write a triangle as

$$\mathcal{J}_{1,2}\mathcal{J}_{2,4}\mathcal{J}_{4,6}(\mathcal{V}_2 \uplus \mathcal{V}_2 \uplus \mathcal{V}_2)$$

or equivalently

$$\mathcal{J}_{1,2}\mathcal{J}_{1,3}(\mathcal{V}_2 \uplus \mathcal{J}_{1,3}(\mathcal{V}_2 \uplus \mathcal{V}_2)).$$

These gadgets are isomorphic but they are not identical. We consider gadgets identical only when they are created by exactly the same sequence of elementary joins from the same base gadgets.

The elementary joins as defined above are defined only on gadgets of some size. This would lead to a partial algebra. To avoid this we extend the gadget algebra by a special object `None` and extend the elementary joins to all gadgets but returning `None` on the new elements of their domains. To simplify the notation, we denote $\mathcal{G}' = \mathcal{G}^* \uplus \{\texttt{None}\}$. More precisely:

Definition 5.2 (Extension of elementary joins)**.** *We extend any elementary join* $\mathcal{J} : \prod_{i\in[n]} \mathcal{G}^{k_i} \to \mathcal{G}^r$ *to a mapping* $\mathcal{J} : \prod_{i\in[n]} \mathcal{G}' \to \mathcal{G}'$ *such that* $\mathcal{J}(x) = \texttt{None}$ *for all* $x \in \prod_{i\in[n]}(\mathcal{G}' \setminus \mathcal{G}^{k_i})$.

This yields the same results as using the partial algebras with the strong homomorphisms as described in Grätzer [2008]. We also use Grätzer [2008] as our main reference for algebraic notions and theorems.

Definition 5.3 (Gadget Algebra)**.** *The* gadget algebra $\mathbb{G}$ *is an algebra whose objects are* $\mathcal{G}' = \mathcal{G}^* \uplus \{\texttt{None}\}$ *and operations are the constant* `None` *and all the elementary joins* $\mathcal{J} : \mathcal{G}'^n \to \mathcal{G}'$.

To simplify the notation, we define a *join* (denoted $\mathcal{J} : \prod_{i\in[n]} \mathcal{G}^{k_i} \to \mathcal{G}^r$ where n is the number of input gadgets, k_i their sizes and r size of the resulting gadget) to be a function created by composing elementary joins such that each input gadget appears in the result exactly once (e.g., $\mathcal{J} : \mathcal{G}^5 \to \mathcal{G}^{10}$ defined by $g \mapsto g \uplus^{5,5} g$ and

$\mathcal{J}' : \mathcal{G}^3 \times \mathcal{G}^4 \to \mathcal{G}^3$ defined by $(g_1, g_2) \mapsto g_1$ are not joins). Note that we explicitly allow the empty gadget so we can reduce any join (even unary one) to a binary join. Also conversely we can replace any sequence of joins by a single join. One class of joins of special interest are *gluings*

$$\mathcal{J}_g^k : \mathcal{G}^k \times \mathcal{G}^k \to \mathcal{G}$$

which join the half-edges with the same labels creating a graph.

Observation 5.4. *Let $\mathcal{J}$ be a join. Then for all $i \in [n]$ there exist joins $\mathcal{J}'$ and $\mathcal{J}''$ such that*

$$\mathcal{J}(g_1, \ldots, g_i, \ldots, g_n) \cong \mathcal{J}'(g_i, \mathcal{J}''(g_1, \ldots, g_{i-1}, g_{i+1}, \ldots, g_n)).$$

Let $\mathcal{J}_1, \mathcal{J}_2$ be joins then there exists join $\mathcal{J}$ such that

$$\mathcal{J}_1(g_1, \mathcal{J}_2(g_2, \ldots, g_n)) \cong \mathcal{J}(g_1, g_2, \ldots, g_n).$$

Moreover if we allow free edges then for any binary join $\mathcal{J}$ with range $\mathcal{G}^0 \uplus \{\texttt{None}\}$ and any gadget g_2 there exists a gadget g_2' such that for all gadgets g_1:

$$\mathcal{J}(g_1, g_2) \cong \mathcal{J}_g(g_1, g_2').$$

Proof. The first two parts are obvious. The moreover part: If $\mathcal{J}$ joins half-edges of g_2 together, we just join them. If $\mathcal{J}$ joins a half-edge of g_1 with a half-edge of g_2, we permute half-edges of g_2 accordingly. Finally if $\mathcal{J}$ joins a half-edge of g_1 with another half-edge of g_1, we add a free edge into g_2. □

With the definitions of gadgets and joins, we can move to graph parameters. The graph parameter is a function on graphs which gives the same values for isomorphic graphs:

Definition 5.5. *A function $\mathcal{P} : \mathcal{G} \to R$ such that $G_1 \cong G_2 \Rightarrow \mathcal{P}(G_1) = \mathcal{P}(G_2)$ is called a* graph parameter. *The set R is the range of this parameter.*

The range of a graph parameter is usually (real) numbers but we do not require it. Note that if needed – like in the case of the parameter which calculates the core of the graph[2] – we can make the theoretical parameter return an isomorphism class (element of $\mathcal{G}/\cong$ in this case) instead of a graph and let the implementation return any member of the class as its representative.

5.2 Decomposable Representations

We want to enrich the descriptions of graph parameters so they capture how the parameters are computed along the gadget decomposition. We will model this as algebras: We first define the gadget algebra where operations are joins and then describe the parameters using homomorphisms from the gadget algebra. The relation of a decomposable representation to its parameter is shown in Figure 5.1.

[2]The core of a graph G is the smallest graph C such that there exists a homomorphism from G into C. The core is determined uniquely up to isomorphism. For more details see, e.g., Hell and Nešetřil [1992].

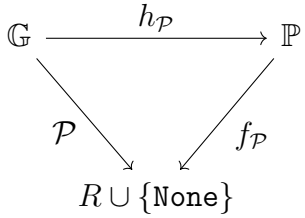

Figure 5.1: Commutative diagram of a decomposable representation

Definition 5.6 (Decomposable Representation)**.** *The* decomposable representation $\mathcal{P}$ *consists of*

- *the* representation algebra $\mathbb{P}$ *of the same type as the gadget algebra* $\mathbb{G}$*, we denote* $\mathcal{J}_{\mathcal{P}}$ *the operation corresponding to the join* $\mathcal{J}$ *and* $\texttt{None}_{\mathcal{P}}$ *the operation corresponding to* `None`*,*
- *an algebra homomorphism* $h_{\mathcal{P}} : \mathbb{G} \to \mathbb{P}$*, and*
- *a mapping* $f_{\mathcal{P}} : \mathbb{P} \to R \cup \{\texttt{None}\}$

such that `None` *is a poison value,*[3] *i.e., any operation which has* `None` *as any argument returns* `None` *and* $f_{\mathcal{P}}(x) = \texttt{None}$ *for all* $x \in \mathcal{G}' \setminus \mathcal{G}^0$*.*

We say that $\mathcal{P}$ *is the decomposable representation of the graph parameter with range* R *defined by* $g \mapsto f_{\mathcal{P}}(h_{\mathcal{P}}(g))$ *and we use* $\mathcal{P}$ *to denote this graph parameter.*

Obviously any parameter can be made into a decomposable representation by setting $\mathbb{P} = \mathbb{G}$, $h_{\mathcal{P}}$ to identity and $f_{\mathcal{P}} = \mathcal{P}$. We call this decomposable representation the *naive representation* for given graph parameter $\mathcal{P}$.

On the other hand some representations are better than others – the intuition is that the homomorphism of algebras $h_{\mathcal{P}}$ is a way to forget the details about the gadgets not important for given graph parameter and the more it forgets the better representation it gives. To determine which representation is better we introduce a notion of homomorphisms of the representations:

Definition 5.7 (Homomorphism)**.** *Let* $\mathcal{P}$ *and* $\mathcal{P}'$ *be decomposable representations of the same graph parameter. A mapping* $h : \mathcal{P} \to \mathcal{P}'$ *is a* homomorphism *from* $\mathcal{P}$ *to* $\mathcal{P}'$ *if the following holds:*

1. h *is a homomorphism* $\mathbb{P} \to \mathbb{P}'$*, and*
2. $f_{\mathcal{P}}(x) = f_{\mathcal{P}'}(h(x))$ *for every* $x \in \mathbb{P}$*.*

Observe that the mapping $h_{\mathcal{P}}$ can be viewed as a homomorphism from the naive representation. Also the intuition that $h : \mathcal{P} \to \mathcal{P}'$ witnesses that $\mathcal{P}'$ is the (non-strictly) better representation is not entirely correct. The homomorphism ensures that $h[\mathcal{P}]$ is better than $\mathcal{P}$ but if h is not onto, $\mathcal{P}'$ contains other objects which might make it more complicated than $\mathcal{P}$. The theoretical solution obviously is to take $\mathcal{P}'$ as small as possible but it might not be simple to do in practice.

Like any other homomorphism of algebras, every homomorphism of decomposable representations h defines a congruence relation $\sim_h$ on objects of its domain

[3] The homomorphism $h_{\mathcal{P}}$ ensures that `None` is a poison value in the image $h_{\mathcal{P}}[\mathbb{G}]$.

via formula $x \sim_h y \Leftrightarrow h(x) = h(y)$. Vice versa, given a congruence $\sim$ on a decomposable algebra of representation $\mathcal{P}$ which respects $f_{\mathcal{P}}$, we can construct a quotient representation $\mathcal{P}/\sim$ and a homomorphism $h : \mathcal{P} \to \mathcal{P}/\sim$.

Definition 5.8 (Congruence)**.** *Let $\mathcal{P}$ be a decomposable representation. We say that a congruence $\sim$ on $\mathbb{P}$ is a* congruence on $\mathcal{P}$ *if for all $x, y \in \mathbb{P}$ it holds $x \sim y \Rightarrow f_{\mathcal{P}}(x) = f_{\mathcal{P}}(y)$.*

Observation 5.9. *Let $\mathcal{P}$ be a decomposable representation, $\sim$ a congruence on it and g_1 and g_2 two gadgets of different size. If $g_1 \sim g_2$ then an equivalence $\sim'$ created from $\sim$ by joining $[g_1]_\sim$ and $[\texttt{None}]_\sim$ is also a congruence.*

Proof. It holds that $f_{\mathcal{P}}([g_1]_\sim) = \texttt{None}$ (at least one of g_1 and g_2 is not a graph) so by joining $[g_1]_\sim$ and $[\texttt{None}]_\sim$ the condition on $f_{\mathcal{P}}$ is preserved. The other condition is

$$\mathcal{J}(x_1, \ldots, x_k, [g_1]_\sim, y_1, \ldots, y_l) = \mathcal{J}(x_1, \ldots, x_k, [\texttt{None}]_\sim, y_1, \ldots, y_l).$$

The right hand side is always `None`. Observe that, due to Definition 5.2, $\mathcal{J}(\ldots, [g_1]_\sim, \ldots) = \texttt{None}$ for all the joins and all the choices of other parameters because always either g_1 or g_2 have the wrong size for the given join. So the second condition is also satisfied. □

Definition 5.10 (Quotient Representation)**.** *Let $\mathcal{P}$ be a decomposable representation and $\sim$ a congruence on it. The* quotient representation *is defined in the following way:*

1. *The algebra is the quotient algebra $\mathbb{P}/\sim$.*
2. *The gadget function is $h_{\mathcal{P}/\sim}(g) = [h_{\mathcal{P}}(g)]_\sim$.*
3. *The final function is $f_{\mathcal{P}/\sim}([x]_\sim) = f_{\mathcal{P}}(x)$.*

The function $h : \mathcal{P} \to \mathcal{P}/\sim$ defined $h(x) = [x]_\sim$ is a homomorphism of decomposable representations. We show that the congruences of a decomposable representation are a complete sublattice[4] of the congruences of its algebra:

Observation 5.11. *Let $\sim_i$ for $i \in I$ be a nonempty system of congruence relations on decomposable representation $\mathcal{P}$. Then both a common coarsening $\bigvee_{i \in I} \sim_i$ and a common refinement $\bigwedge_{i \in I} \sim_i$ exist and they are congruences on $\mathcal{P}$.*

Proof. Both $\bigvee_{i \in I} \sim_i$ and $\bigwedge_{i \in I} \sim_i$ exist as congruences on $\mathbb{P}$ because all congruences on an algebra are a complete lattice. It remains to show that they are also congruences on $\mathcal{P}$. It is trivial for $\bigwedge_{i \in I} \sim_i$ because it is a refinement of each of the $\sim_i$. For $\sim = \bigvee_{i \in I} \sim_i$ we use the fact that $x \sim y$ if and only if there exists a sequence of objects $o_0, \ldots, o_k$ and a sequence of indices $i_1, \ldots, i_k \in I$ such that

$$x = o_0 \sim_{i_1} o_1 \sim_{i_2} \cdots \sim_{i_k} o_k = y.$$

Hence if $x \sim y$ but $f_{\mathcal{P}}(x) \neq f_{\mathcal{P}}(y)$ then $f_{\mathcal{P}}(o_i) \neq f_{\mathcal{P}}(o_{i+1})$ for some i as well. □

[4] A lattice L is complete if $\bigvee X$ and $\bigwedge X$ exist for every set $X \subseteq L$. For the details see, e.g., Grätzer [2008].

Restricting ourselves to the quotient representations, the natural question is whether there exists the best possible representation. The answer is positive and it is a direct consequence of Observation 5.11.

Observation 5.12 (The Best Representation)**.** *For every graph parameter $\mathcal{P}$ there exists a unique maximal congruence $\sim$ on its naive representation $\mathcal{N}$. We call $\mathcal{N}/\sim$ the* best decomposable representation *of $\mathcal{P}$.*

Proof. Consider the set of all congruencies on $\mathcal{N}$ and apply Observation 5.11. □

5.3 Linear Representations

So the best decomposable representation exists but it is rarely useful in practice. The main reason is that it does not have any additional structure which would enable us to do computations in it efficiently and it usually is still infinite. To remedy this, we will require representations to have some additional structure and require that the homomorphisms preserve it.

The structure we choose is a vector space and linear functions. We choose vector spaces because they are very easy to work with and yet they are expressive enough with a lot of tools to use during the analysis. We usually work over real numbers but the theory works for any other field of characteristic zero.

So we define a linear representation, a naive linear representation, etc., and finish by showing that there exists the best linear representation. The main difference is that there usually exists a linear representation such that the subspace generated by gadgets of size k has finite dimension for all k. We call such representations finite.

Definition 5.13 (Linear Representation)**.** *A decomposable representation $\mathcal{P}$ with range R together with sets $B_\mathcal{P}$ and $\{e_b : b \in B_\mathcal{P}\}$ is a* linear representation *over a field F if*

1. *F has characteristic zero,*[5]
2. *R is a vector space over F,*
3. *elements of $\mathbb{P}$ form a vector space over F,*
4. *$h_\mathcal{P}(\texttt{None})$ is the zero vector,*
5. *all the functions $\mathcal{J}_\mathcal{P}$ are linear in all their arguments,*
6. *$f_\mathcal{P}$ is also a linear function, and*
7. *the set $\{e_b : b \in B_\mathcal{P}\}$ is a basis of $\mathbb{P}$ indexed by $B_\mathcal{P}$ so that we can define support* $\operatorname{supp}(v) \subset B_\mathcal{P}$*. We call elements of the set*

$$B^k_\mathcal{P} = \bigcup\{\operatorname{supp}(h_\mathcal{P}(g)) : g \in \mathcal{G}^k\}$$

the k-boundaries *and we call the elements of $O^k_\mathcal{P} = F^{B^k}$ (as a subspace of $\mathbb{P}$) k-*multiplicity vectors*. We choose the basis e_b such that it satisfies $\operatorname{supp}(h_\mathcal{P}(g_1)) \cap \operatorname{supp}(h_\mathcal{P}(g_2)) = \emptyset$ for all gadgets g_1, g_2 of different sizes. If such basis does not exist, we call the representation* improper*, otherwise it is* proper.[6]

[5] This is not strictly required for this definition but all our theorems assume this.

[6] We show in Observation 5.15 that we can restrict ourselves to proper representations.

We say that the representation is finite *if all the sets B^k are finite. We omit the subscript $\mathcal{P}$ whenever possible.*

We define homomorphisms *and* congruences *of linear representations the same way as for the decomposable representations (Definitions 5.7 and 5.8) but moreover we require that they respect the structure of the vector space (formally, we extend the parameter algebra with the vector space operations).*

We want to show that we can ignore improper representations – i.e., that there is always a better representation which is proper. It is a generalization of Observation 5.9. But to do that we need to observe that congruences on linear representations form a complete lattice.

Observation 5.14. *Let $\mathcal{P}$ be a linear representation. Then the set $\mathcal{C}$ of all congruences on $\mathcal{P}$ is a complete lattice – i.e., for any $C \subseteq \mathcal{C}$ there exists both a congruence which is common refinement and a congruence which is a common coarsening of the congruences in C.*

Proof. We apply the proof of the Observation 5.11. (We cannot apply the observation directly because our algebra is extended by operation + on its elements and by multiplication by a scalar value.) □

Observation 5.15. *Let $\mathcal{P}$ be an improper linear representation. Then there exists a congruence $\sim$ on $\mathcal{P}$ such that $\mathcal{P}' = \mathcal{P}/\sim$ is a proper linear representation.*

Proof. For each $k \in \mathbb{N}$ consider $I_k = \langle h_{\mathcal{P}}(g) : g \in \mathcal{G}^k \rangle$ and define

$$N' = \bigcup_{i \neq j \in \mathbb{N}} (I_i \cap I_j), \qquad N = \langle N' \rangle.$$

Note that if $N = \{0\}$, the representation is proper. We define $\sim$ via formula $x \sim y \Leftrightarrow x - y \in N$. We need to prove that $\sim$ is a congruence. If it is, the representation $\mathcal{P}'$ is proper from the definition of $\sim$. Obviously $\sim$ is a congruence in the sense of vector spaces because N is a linear subspace of $\mathbb{P}$. It remains to show that it respects $f_{\mathcal{P}}$ and all the (images of) joins $\mathcal{J}_{\mathcal{P}}$.

Consider any $z \in N'$. By definition $z \in I_i \cap I_j$ (where $i \neq j$). At least one of i, j is not zero, hence $f_{\mathcal{P}}(z) = 0$ ($f_{\mathcal{P}}$ can be non zero only for graphs due to Definition 5.6). Now consider $v = J_{\mathcal{P}}(g_1, \ldots, g_k, z, g_{k+2}, \ldots, g_{k'})$ for any join $\mathcal{J}$ and gadgets g_i. Again at least one of i and j is the wrong size of the gadget for $(k+1)$-th argument of the join $\mathcal{J}$, so $v = 0$ due to Definition 5.2. Because N is the span of N', the same holds for all elements of N. Hence $\sim$ respects $f_{\mathcal{P}}$ and all $\mathcal{J}_{\mathcal{P}}$. □

Hence from now on we consider only proper linear representations unless explicitly noted otherwise. The definition of k-boundaries as non-zero indices of the vector space may seem unnatural at first with the better definition being (the span of) the image of $\mathcal{G}^k$. We choose the former definition because in general we are not able to calculate $\langle h_{\mathcal{P}}[\mathcal{G}^k] \rangle$. If $\langle h_{\mathcal{P}}[\mathcal{G}^k] \rangle \subsetneq \langle e_b : b \in B^k \rangle$, we can replace $\langle e_b : b \in B^k \rangle$ with $\langle h_{\mathcal{P}}[\mathcal{G}^k] \rangle$ obtaining a better representation. Note that given a homomorphism $m : \mathcal{P} \to \mathcal{P}'$ the formula $m[O^k_{\mathcal{P}}] \subseteq O^k_{\mathcal{P}'}$ does not hold in general but only $m[\langle f_{\mathcal{P}}(g) : g \in \mathcal{G}^k \rangle] \subseteq O^k_{\mathcal{P}'}$ does.

Definition 5.16 (Naive Linear Representation)**.** *The* naive linear representation *$\mathcal{S}_{\mathcal{P}}$ of a graph parameter $\mathcal{P} : \mathcal{G} \to \mathbb{R}$ is defined:*

1. $F = \mathbb{R}$, $R = \mathbb{R}$,

2. $h_{\mathcal{S}_\mathcal{P}}(g) = e_g$ *for all* $g \in \mathcal{G}^*$ *(to simplify the notation we will treat* g *as* e_g*),*

3. $\mathbb{S}_\mathcal{P}$ *is a space generated by* $h_{\mathcal{S}_\mathcal{P}}[\mathcal{G}^*] \subseteq \mathbb{R}^{\mathcal{G}^*}$ *(note that this is a strict subset of* $\mathbb{R}^{\mathcal{G}^*}$ *as we allow only finitely many non-zero coordinates),*

4. $\mathcal{J}_{\mathcal{S}_\mathcal{P}}$ *is the linear extension of* $\mathcal{J}$ *for all joins* $\mathcal{J}$ *(using the simplification mentioned in Item 2), and*

5. $f_{\mathcal{S}_\mathcal{P}}$ *is the linear extension of* $\mathcal{P}$*.*

Observation 5.17. *Let* $\mathcal{P}$ *be any linear representation. Then* $h_\mathcal{P}$ *can be uniquely extended into a homomorphism from* $\mathcal{S}_\mathcal{P}$ *into* $\mathcal{P}$*.*

Proof. Obvious. □

We call this extension also $h_\mathcal{P}$. Note that $h_\mathcal{P}$ (as a homomorphism from $\mathcal{S}_\mathcal{P}$) might not be onto unless we constructed $\mathcal{P}$ as a quotient of $\mathcal{S}_\mathcal{P}$.

Theorem 5.18 (The Best Linear Representation)**.** *For every graph parameter* $\mathcal{P}$ *there exists a unique maximal congruence* $\sim$ *on its naive linear representation* $\mathcal{S}$*. We call* $\mathcal{S}/\!\sim$ *the* best linear representation *of* $\mathcal{P}$*, we denote it* $\mathcal{L}_\mathcal{P}$*, and it is a proper representation.*

Proof. We consider all congruences on $\mathcal{S}$ and we apply Observation 5.14 to get a common coarsening $\sim$. Due to Observation 5.15, $\mathcal{S}/\!\sim$ is a proper linear representation. □

Note that we did not specify the basis of the best representation. This is because any choice will work equally well.

Observation 5.19. *Let* $\mathcal{P}$ *be a proper representation which is also a quotient of its naive linear representation* $\mathcal{S}$*. Then* $|B^k_\mathcal{P}| = \dim h_\mathcal{P}[O^k_\mathcal{S}]$ *for every choice of the fixed basis in* $\mathcal{P}$*.*

Proof. The naive representation has a basis consisting of images of gadgets (so $O^k_\mathcal{P} \supseteq h_\mathcal{P}[O^k_\mathcal{S}]$) and the whole $\mathbb{P}$ is a direct product of $\{h_\mathcal{P}[O^k_\mathcal{S}]\}_k$ (so $O^k_\mathcal{P} \subseteq h_\mathcal{P}[O^k_\mathcal{S}]$). Hence $O^k_\mathcal{P} = h_\mathcal{P}[O^k_\mathcal{S}]$ and $|B^k_\mathcal{P}| = \dim O^k_\mathcal{P} = \dim h_\mathcal{P}[O^k_\mathcal{S}]$. □

There also exists a homomorphism from the best decomposable representation into the best linear representation. To prove this we need the following well known fact about algebras:

Fact 5.20. *Let* $h : \mathbb{A}_1 \to \mathbb{A}_2$ *be a homomorphism of algebras which is also a bijection. Then* h *is an isomorphism.*

Observation 5.21. *There exists a homomorphism (in the decomposable representation sense) from the best decomposable representation* $\mathcal{B}$ *into the best linear representation* $\mathcal{L}$*.*

Proof. By definition there exists an onto homomorphism $j : h_{\mathcal{L}}[\mathcal{G}^*] \to \mathcal{B}$. We show that j is injective. For contradiction assume that there exist gadgets g_1, g_2 such that $h_{\mathcal{L}}(g_1) \neq h_{\mathcal{L}}(g_2)$ but $h_{\mathcal{B}}(g_1) = h_{\mathcal{B}}(g_2)$. We can take $\mathcal{B}$ and turn it into a linear representation $\mathcal{B}'$ the same way we turned the naive decomposable representation into the naive linear representation. It holds $h_{\mathcal{B}'}(g_1) = h_{\mathcal{B}'}(g_2)$ which is a contradiction with $\mathcal{L}$ being the best linear representation.

So j is an bijective homomorphism and by Fact 5.20 it is a isomorphism. Hence there exists a homomorphism $h : \mathcal{B} \to h_{\mathcal{L}}[\mathcal{G}^*] \subset \mathcal{L}$ defined $h = j^{-1}$. □

In practice it is not enough that the representation is finite but we are very interested how fast $|B^k|$ grows. We show that the best linear representation is also the one with $|B^k|$ growing as slow as possible.

Observation 5.22. *The best linear representation has the smallest number of k-boundaries among all linear representations of the given parameter.*

Proof. By contradiction. Assume that some other representation $\mathcal{P}$ has less k-boundaries. The function $h_{\mathcal{P}}$ defines congruence $\sim_{\mathcal{P}}$ on the naive linear representation $\mathcal{S}$. The congruence $\sim$ used to construct the best representation is coarser than all other congruences on $\mathcal{S}$, specially $\sim_{\mathcal{P}}$. Hence the best linear representation must have at most as many k-boundaries as $\mathcal{P}$. Contradiction. □

The important property of the best representation is that there exists a homomorphism into it from any other quotient of the naive representation. So we can take a quotient representation and either prove that it is best or improve it and get closer to the best one. We can never do a "wrong" move which would leave us with non-optimal representation and no way to improve it. To prove that a given representation is best, it is enough to show that the size of each B^k is as small as possible.

Observation 5.23. *Let $\mathcal{P}$ be a linear representation and let $\mathcal{L}$ be the best linear representation of the same parameter. If $|B^k_{\mathcal{P}}| = |B^k_{\mathcal{L}}| < \infty$ for all k then $\mathcal{P}$ is isomorphic to $\mathcal{L}$.*

Proof. Because the $|B^k_{\mathcal{P}}|$ are as small as possible, $h_{\mathcal{P}}$ as a homomorphism from the naive representation is onto. By definition of $\mathcal{L}$ there is a homomorphism $m : h_{\mathcal{P}}[\mathcal{S}] \to \mathcal{L}$ from image of the naive representation (i.e., the whole $\mathcal{P}$ in this case) into $\mathcal{L}$ which is also onto. Homomorphism m must be injective because $\mathcal{P}$ and $\mathcal{L}$ have the same number of k-boundaries – i.e., the corresponding vector spaces have the same finite dimension. Hence m is a bijective homomorphism, and by Fact 5.20 it is an isomorphism. □

A way to show this equality is to construct for every k a matrix A^k indexed by k-gadgets defined

$$A^k_{g_1,g_2} = \mathcal{P}(\mathcal{J}_g(g_1, g_2))$$

and show that is has rank $|B^k|$.

Observation 5.24. *Let $\mathcal{P}$ be a linear representation and let S be some set of k-gadgets. Define matrix A^k by $A^k_{g_1,g_2} = \mathcal{P}(\mathcal{J}^k_g(g_1, g_2))$ where $g_1, g_2 \in S$. Then $\operatorname{rank}(A^k) \leq |B^k_{\mathcal{P}'}|$ for every linear representation $\mathcal{P}'$ of the same graph parameter.*

Proof. The matrix A^k can also be written as $A^k = M_k^T G M_k$ where G is the matrix of the bilinear map $f_{\mathcal{P}}((\mathcal{J}_g^k)_{\mathcal{P}}(.,.))$ and M_k is a matrix whose columns are multiplicity vectors (i.e., $h_{\mathcal{P}}(.)$) of the k-gadgets in S treated as elements of F^{B^k}. Because the rank of the result of matrix multiplication is at most the minimum of the ranks of the involved matrices, this gives a lower bound on the rank of the matrix M_k (and also G) for any representation of this parameter. Hence it bounds the number of its rows from below and each row corresponds to a k-boundary. □

We want to observe that if two gadgets can be distinguished by a parameter then they are distinguished by gluing. This will allow us to examine only gluing and not all possible joins involving gadgets of the given size. This is unfortunately true only if we allow free edges. Note that by Observation 5.4 any sequence of joins can be reduced to a single join.

Observation 5.25. *Let $\mathcal{P}$ be a linear representation allowing free edges and let g_1, g_2 be k-gadgets. If there exists any join $\mathcal{J}$ and gadgets $f_1, \ldots, f_n$ such that*

$$\mathcal{P}(\mathcal{J}(g_1, f_1, \ldots, f_n)) \neq \mathcal{P}(\mathcal{J}(g_2, f_1, \ldots, f_n))$$

then there exists a k-gadget f such that $\mathcal{P}(\mathcal{J}_g(g_1, f)) \neq \mathcal{P}(\mathcal{J}_g(g_2, f))$.

Proof. Obvious. Just join all the other gadgets first by Observation 5.4. □

This immediately gives us that any such linear representation can be improved by decreasing the number of k-boundaries to the rank of matrix G from the proof of Observation 5.24.

Observation 5.26. *Let $\mathcal{P}$ be a linear representation allowing free edges and let $n \in \mathbb{N}$. Then there exists a linear representation $\mathcal{P}'$ which has the same k-boundaries as $\mathcal{P}$ for all $k \neq n$ and has $\operatorname{rank}(A^n)$ of n-boundaries where A^n is a matrix with entries $A^n_{i,j} = \mathcal{P}(\mathcal{J}_g(e_i, e_j))$ and $\{e_i : i \in I\}$ is any basis of $O^n_{\mathcal{P}}$.*

Proof. Let $I' \subset I$ be maximal such that the corresponding columns (or equivalently rows because A^n is symmetric) of A^n are independent. (So $|I'| = \operatorname{rank}(A^n)$.) Let $D = I \setminus I'$ and for $d \in D$ let $c_i^d \in \mathbb{R}$ be such that

$$A_{*,d} = \sum_{i \in I'} c_i^d A_{*,i}.$$

Define vectors y^d such that $y_i^d = c_i^d$ for $i \in I'$, $y_d^d = -1$ and $y_{d'}^d = 0$ for all other $d' \in D$. Obviously $y^d \in \ker(A^n)$. Hence we can transform any multiplicity vector $v \in O^n$ to

$$v' = v + \sum_{d \in D} v_d \, y^d.$$

Then it holds $(\mathcal{J}_g)_{\mathcal{P}}(v, w) = (\mathcal{J}_g)_{\mathcal{P}}(v', w)$ for all $w \in O^n$ and $(v')_d = 0$ for all $d \in D$. So we can just drop the coordinates D which gives us the representation $\mathcal{P}'$. □

Hence the rank of matrix A^k is an upper bound on the number of k-boundaries of the best representation. For an example of a representation with a full-rank matrix A^k which is not the best representation see the representation of k-edge colorings described in Section 5.6. On the other hand we show that our bounds meet for the best representation. To do this we need to observe that the best representation has a basis consisting of images of gadgets.

Observation 5.27. *Let $\mathcal{P}$ be a linear representation which is a quotient of the corresponding naive linear representation $\mathcal{S}$. Then there exist k-gadgets $\{g_i\}_i$ such that $\{h_{\mathcal{P}}(g_i)\}_i$ is a basis of $O^k_{\mathcal{P}}$.*

Proof. The naive representation has basis consisting of images of gadgets, hence so does any its quotient. □

Corollary 5.28. *Let $\mathcal{L}$ be the best linear representation which is also finite and allows free edges. Let $\{g_i\}_i$ be set of k-gadgets such that $\{h_{\mathcal{L}}(g_i)\}_i$ is a basis of $O^k_{\mathcal{L}}$. Then $|B^k_{\mathcal{L}}| = \operatorname{rank}(A^k)$ where A^k is a matrix defined $A^k_{i,j} = \mathcal{L}(\mathcal{J}^k_g(g_i, g_j))$.*

In the examples we present below, it is usually the case that the gadgets are mapped to non-negative vectors. We show that this is not a coincidence but a natural property of a linear representation of a non-negative graph parameter. First we need to generalize the notion of non-negative vectors to account for different rotations:

Definition 5.29 (Cone)**.** *A* cone *is a subset of a linear space over $\mathbb{R}$ which is closed under linear combinations with non-negative coefficients. A cone is* proper *if it does not contain any line (i.e., a set of form $\{\alpha v : \alpha \in \mathbb{R}\}$ for a fixed non-zero vector v). A cone* generated *by set X is the set of all non-negative linear combinations of the elements of X.*

Observation 5.30. *Let $\mathcal{P}$ be a linear representation over $\mathbb{R}$ such that the values of graphs $\{\mathcal{P}(G) : \forall G \in \mathcal{G}\}$ generate a proper cone V. Consider cone C generated by the images of gadgets $\{h_{\mathcal{P}}(g) : \forall g \in \mathcal{G}^*\}$. If $\{\alpha x : \forall \alpha \in \mathbb{R}\} \subset C$, then $f_{\mathcal{P}}(\mathcal{J}_{\mathcal{P}}(x, h_{\mathcal{P}}(g))) = 0$ for all gadgets g and all binary joins $\mathcal{J}$.*

Proof. For contradiction let us assume that $f_{\mathcal{P}}(\mathcal{J}_{\mathcal{P}}(x, h_{\mathcal{P}}(g))) \neq 0$ for some join $\mathcal{J}$, some gadget g and some x such that $\{\alpha x : \forall \alpha \in \mathbb{R}\} \subset C$. Fix $\beta \in \mathbb{R}$ such that

$$f_{\mathcal{P}}(\mathcal{J}_{\mathcal{P}}(\beta x, h_{\mathcal{P}}(g))) \notin V.$$

Such β exists because V is a proper cone. We can write βx as a non-negative linear combination of images of gadgets and then use linearity:

$$\begin{aligned} \beta x &= \sum_i \alpha_i h_{\mathcal{P}}(g_i) \\ f_{\mathcal{P}}(\mathcal{J}_{\mathcal{P}}(\beta x, h_{\mathcal{P}}(g))) &= f_{\mathcal{P}}(\mathcal{J}_{\mathcal{P}}(\sum_i \alpha_i h_{\mathcal{P}}(g_i), h_{\mathcal{P}}(g))) \\ &= \sum_i \alpha_i f_{\mathcal{P}}(\mathcal{J}_{\mathcal{P}}(h_{\mathcal{P}}(g_i), h_{\mathcal{P}}(g))) \end{aligned}$$

Note that if the result of the join $\mathcal{J}(g_i, g)$ is not a graph, we have

$$f_{\mathcal{P}}(\mathcal{J}_{\mathcal{P}}(h_{\mathcal{P}}(g_i), h_{\mathcal{P}}(g))) = 0.$$

So we have

$$\sum_i \alpha_i \mathcal{P}(\mathcal{J}(g_i, g)) \notin V$$

but all $\alpha_i \geq 0$ and $\mathcal{P}(\mathcal{J}(g_i, g)) \in V$ by definition. Contradiction. □

Using this observation and applying a suitable linear transformation we get:

Corollary 5.31. *Let $\mathcal{P}$ be a non-negative real-valued graph parameter. Then is has a best representation over $\mathbb{R}$ in which all gadgets are mapped to non-negative vectors.*

An interesting question is the characterization of the graphs parameters which do have a finite linear representation. We leave it as an open problem for further research.

Problem 5.32. *Characterize graph parameters which have a finite linear representation over $\mathbb{Q}$, $\mathbb{R}$ or $\mathbb{C}$. What if we restrict growth of $|B^k|$?*

5.4 Graph Sequences

We conclude this section with definition of graph sequences created by repeatedly joining with the same gadget which will help us analyze asymptotic behavior of the linear representations.

Definition 5.33 (Graph Sequence)**.** *A* graph sequence $(G_i)_{i\geq z}$ *is defined by its* initial index z, initial gadget g_{init}, step gadget g_{step}, step join $\mathcal{J}_{\text{step}}$ *and* final join $\mathcal{J}_{\text{fin}}$ *by the following formulas:*

$$g_z = g_{\text{init}}, \qquad g_{i+1} = \mathcal{J}_{\text{step}}(g_{\text{step}}, g_i), \qquad G_i = \mathcal{J}_{\text{fin}}(g_i).$$

Note that sizes of the initial gadget and the result of the step join must be the same and we call this value *size of the sequence*, denoted s. For example cyclic ladders (also know as prizm graphs) are a graph sequence of size 2 with

$$g_{\text{step}} = \pi[1,3,4,2](\mathcal{J}_{3,4}(\mathcal{V}_3 \uplus \mathcal{V}_3)), \qquad \mathcal{J}_{\text{step}}(g_a, g_b) = \mathcal{J}_{3,4}\mathcal{J}_{3,5}(g_a \uplus g_b),$$
$$z = 2, \qquad g_{\text{init}} = \mathcal{J}_{\text{step}}(g_{\text{step}}, g_{\text{step}}), \qquad \mathcal{J}_{\text{fin}} = \mathcal{J}_{1,2} \cdot \mathcal{J}_{2,3}.$$

We can also obtain Möbius ladder by changing the final join to $\mathcal{J}_{\text{fin}} = \mathcal{J}_{1,2} \cdot \mathcal{J}_{1,3}$. For illustration see Figure 5.2.

The following observation shows that asymptotic behavior of a linear representation of a graph sequence can be determined by analyzing a single matrix. Moreover this matrix has finite size if the representation is finite.

Observation 5.34. *Let $\mathcal{P}$ be a linear representation over F with range F and let $(G_i)_{i\geq z}$ be a graph sequence of size s. Then value $\mathcal{P}(G_i)$ can be expressed as $m_{\text{fin}}^T A^{i-z} m_{\text{init}}$, where $m_{\text{init}} = h_{\mathcal{P}}(g_{\text{init}}) \in F^{B^s}$ is the multiplicity vector of the base gadget, $A = (\mathcal{J}_{\text{step}})_{\mathcal{P}}(h_{\mathcal{P}}(g_{\text{step}}), .) \in F^{B^s \times B^s}$ is the matrix describing addition of the step gadget and $m_{\text{fin}} = f_{\mathcal{P}} \cdot (\mathcal{J}_{\text{fin}})_{\mathcal{P}} \in F^{B^s}$ is the vector describing the final join.*

Proof. Obvious. □

We can also obtain an explicit formula for $\mathcal{P}(G_i)$ by using Jordan normal form (see, e.g., Horn and Johnson [2012]). Note that we have to work over complex numbers as real matrices can have complex eigenvalues.

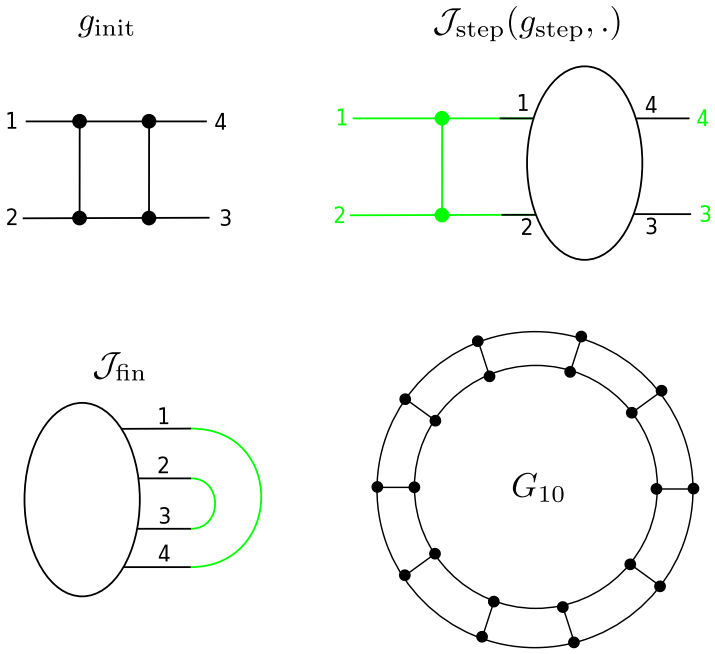

Figure 5.2: Cyclic ladders

Observation 5.35. *Let $u, v \in \mathbb{C}^n$ be vectors and $J \in \mathbb{C}^{n\times n}$ be a matrix such that diagonal entries are $\lambda \in \mathbb{C}$, entries right above the diagonal are ones and all other entries are zeros (i.e., the matrix is a single Jordan block). Then*

$$u^T J^k v = \sum_{j=1}^{n} u_j \sum_{i=0}^{n-j} \binom{k}{i} v_{j+i} \lambda^{k-i} \qquad \forall k \geq n.$$

Proof. We will prove that $(J^k v)_j = \sum_{i=0}^{n-j} \binom{k}{i} v_{j+i} \lambda^{k-i}$ for $k \geq n$. We proceed by the induction on j from n to 1 and k from 1 to ∞. The base case is $j = n$ for any k:

$$(J^k v)_n = \sum_{i=0}^{0} \binom{k}{i} v_{n+i} \lambda^{k-i} = v_n \lambda^k$$

which holds. Induction step from $j+1$ to j for $k \geq n-j$:

$$\begin{aligned}
(J^k v)_j &= (J(J^{k-1}v))_j = \lambda (J^{k-1}v)_j + (J^{k-1}v)_{j+1} \\
&= \lambda \sum_{i=0}^{n-j} \binom{k-1}{i} v_{j+i} \lambda^{k-1-i} + \sum_{i=0}^{n-j-1} \binom{k-1}{i} v_{j+i+1} \lambda^{k-1-i} \\
&= \sum_{i=0}^{n-j} \binom{k-1}{i} v_{j+i} \lambda^{k-i} + \sum_{i=0}^{n-j-1} \binom{k-1}{i} v_{j+i+1} \lambda^{k-1-i} \\
&= v_j \lambda^k + \sum_{i=0}^{n-j-1} \binom{k-1}{i+1} v_{j+i+1} \lambda^{k-i-1} + \sum_{i=0}^{n-j-1} \binom{k-1}{i} v_{j+i+1} \lambda^{k-1-i} \\
&= v_j \lambda^k + \sum_{i=0}^{n-j-1} \left(\binom{k-1}{i+1} + \binom{k-1}{i} \right) v_{j+i+1} \lambda^{k-i-1} \\
&= v_j \lambda^k + \sum_{i=0}^{n-j-1} \binom{k}{i+1} v_{j+i+1} \lambda^{k-(i+1)}
\end{aligned}$$

$$= v_j\lambda^k + \sum_{i=1}^{n-j}\binom{k}{i}v_{j+i}\lambda^{k-i}$$
$$= \sum_{i=0}^{n-j}\binom{k}{i}v_{j+i}\lambda^{k-i}$$ □

Corollary 5.36. *Let $u, v \in \mathbb{C}^n$ be vectors and $M \in \mathbb{C}^{n\times n}$ be a matrix. Then*

$$u^T M^k v = \sum_{i=1}^{l} \lambda_i^k P_i(k) \qquad \forall k \geq n$$

where $\lambda_1, \lambda_2, \ldots, \lambda_l$ are all the distinct eigenvalues of M and P_i is a polynomial of degree at most $a_i - g_i$ where a_i is the algebraic multiplicity of λ_i and g_i its geometric multiplicity.

Proof. We choose P and J such that $M = PJP^{-1}$ and J is a matrix in the Jordan normal form. Then we apply Observation 5.35 to each block of $u^T P$, J and $P^{-1}v$. For the l-th block corresponding to λ_i we obtain a formula $\lambda_i^k Q_{i,l}(k)$ where $Q_{i,l}$ is a polynomial of degree at most the size of the block minus one. The size of the block cannot be larger than $a_i - g_i + 1$ (Section 3.1 of Horn and Johnson [2012]). So $P_i = \sum_l Q_{i,l}$. □

5.5 Linear Representations and Edge Coloring Models

There is a connection between finite linear representations and edge coloring models. We first shortly review edge coloring models using mostly terminology of Szegedy [2007]. Then we prove that we can derive a finite linear representation from every edge coloring model but not vice versa. In praticular, we show that circuit double covers cannot be counted by neither a real valued nor a complex valued edge coloring model.

Szegedy [2007] proved the characterization of the edge coloring models with values in $\mathbb{R}$. Later Draisma et al. [2012] and Schrijver [2015] proved a characterization of complex valued edge coloring models but we will not explore this further. Lets start with definition of the real valued edge coloring model (the complex valued one differs only by replacing $\mathbb{R}$ with $\mathbb{C}$):

Definition 5.37 (Edge Coloring Model)**.** *An $\mathbb{R}$-valued* edge coloring model *is a graph parameter $t : \mathcal{G} \to \mathbb{R}$ given by a function $t : \mathbb{N}^d \to \mathbb{R}$ (where d is the number of colors) and formula*

$$t(G) = \sum_{\psi: E(G)\to[d]} \prod_{v\in V(G)} t(v_\psi)$$

where $v_\psi \in \mathbb{N}^d$ is a vector such that $(v_\psi)_i$ is the number of edges of color i incident to vertex v in coloring ψ (note that loops are counted twice).

We say that a graph parameter $f : \mathcal{G} \to \mathbb{R}$ is multiplicative if $f(G_1 \uplus G_2) = f(G_1)f(G_2)$ for all graphs $G_1, G_2 \in \mathcal{G}$ and $f(E) = 1$ where E is the empty graph. The next property we need is edge reflection positivity and it is easiest to define in terms of quantum graphs:

Definition 5.38 (Quantum Graph and Gadget)**.** *A* quantum graph *is a formal linear combination of graphs. A* quantum gadget *is a formal linear combination of gadgets of the same size.*

Note that gadget joins and any linear representation can be linearly extended to quantum graphs. We remind that $\mathcal{J}_g^k : \mathcal{G}^k \times \mathcal{G}^k \to \mathcal{G}$ (called gluing, defined for all k) is a binary join which joins the half-edges with the same labels. A quantum graph Q is called *edge reflection symmetric* if there exists a quantum gadget H such that $Q = \mathcal{J}_g^{|H|}(H, H)$. We say that a graph parameter $f : \mathcal{G} \to \mathbb{R}$ is *edge reflection positive* if $f(Q) \geq 0$ for all edge reflection symmetric quantum graphs Q. The characterization of the real valued edge coloring models is:

Theorem 5.39 (Szegedy [2007])**.** *A function $f : \mathcal{G} \to \mathbb{R}$ can be realized by a real valued edge coloring model if and only if f is multiplicative and edge reflection positive graph parameter.*

As noted above, every real or complex valued edge coloring model gives us a finite linear representation:

Theorem 5.40. *Let $t : \mathbb{N}^d \to F$ be an F-valued edge coloring model ($F \in \{\mathbb{R}, \mathbb{C}\}$). Then there exists a finite linear representation $\mathcal{P}$ over F defined (again we define all the functions on the basis only):*[7]

1. *range is F,*
2. *$B^k = [d]^k$ (colorings of the half-edges with d colors),*
3. *$h_{\mathcal{P}}(g) = \sum_{b \in B^{|g|}} \mathbf{1}_b \sum_{\psi > b} \prod_{v \in V(g)} t(v_\psi)$,*
4. *$(\mathcal{J}_{i,j})_{\mathcal{P}}(\mathbf{1}_b) = 0$ if $b_i \neq b_j$ and $\mathbf{1}_{b'}$ otherwise where b' is b without the i-th and the j-th component,*
5. *$\uplus_{\mathcal{P}}(\mathbf{1}_b, \mathbf{1}_{b'}) = \mathbf{1}_{b.b'}$ where . means concatenation,*
6. *$\pi[\sigma]_{\mathcal{P}}(\mathbf{1}_b) = \mathbf{1}_{\sigma(b)}$,*
7. *$f_{\mathcal{P}}(m) = m$,*

such that $t(G) = \mathcal{P}(G)$ for all graphs G. It holds that $|B^k| = d^k$.

Proof. Obviously $t = \mathcal{P}$. Also all the required functions are linear. It remains to show that $\mathcal{P}$ is a decomposable representation, i.e., that for all joins $\mathcal{J} : \prod_{i \in [n]} \mathcal{G}^{k_i} \to \mathcal{G}^r$ and for all gadgets of correct sizes $g_1, g_2, \ldots, g_n$ the equation $h_{\mathcal{P}}(\mathcal{J}(g_1, g_2, \ldots, g_n)) = \mathcal{J}_{\mathcal{P}}(h_{\mathcal{P}}(g_1), h_{\mathcal{P}}(g_2), \ldots, h_{\mathcal{P}}(g_n))$ holds.

It is enough to prove it for elementary joins. For $\mathcal{J}_{1,2}$ and a k-gadget g we need to show $h_{\mathcal{P}}(\mathcal{J}_{1,2}^k(g)) = (\mathcal{J}_{1,2}^k)_{\mathcal{P}}(h_{\mathcal{P}}(g))$. The proof of it is not complicated but it is somewhat technical. We need to distinguish to which gadget we apply v_ψ (from the definition of an edge coloring model) so we denote $v_{\psi,g}$ the value of

[7] We denote $\psi > b$ the non-proper edge-coloring ψ extending b which is interpreted as coloring of the half-edges.

v_ψ in gadget g. We start by expanding the left-hand side (the final step is from $\mathcal{J}_{1,2}(g)$ to g):

$$h_{\mathcal{P}}(\mathcal{J}^k_{1,2}(g)) = \sum_{b\in[d]^{k-2}} \mathbf{1}_b \sum_{\psi>b} \prod_{v\in V(\mathcal{J}^k_{1,2}(g))} t(v_{\psi,\mathcal{J}_{1,2}(g)})$$
$$= \sum_{b\in[d]^{k-2}} \mathbf{1}_b \sum_{c\in[d]} \sum_{\psi>(c,c).b} \prod_{v\in V(g)} t(v_{\psi,g})$$

Now we expand the right-hand side (in the last step we split out the colors of the first two half-edges). Expression b' denotes b without the first two coordinates.

$$(\mathcal{J}^k_{1,2})_{\mathcal{P}}(h_{\mathcal{P}}(g)) = \sum_{\substack{b\in B^k \\ b_1=b_2}} (h_{\mathcal{P}}(g))_b \mathbf{1}_{b'}$$
$$= \sum_{\substack{b\in[d]^k \\ b_1=b_2}} \mathbf{1}_{b'} \sum_{\psi>b} \prod_{v\in V(g)} t(v_{\psi,g})$$
$$= \sum_{b\in[d]^{k-2}} \sum_{c\in[d]} \mathbf{1}_b \sum_{\psi>(c,c).b} \prod_{v\in V(g)} t(v_{\psi,g})$$

We obtained the same formula so the equality holds. The proof is similar and even more trivial for $\uplus$ and $\pi[\sigma]$ so we omit it. □

The representation from Theorem 5.40 is a slight generalization of the representation of edge colorings shown below (Section 5.6.2). Also whether this representation is optimal or not depends on the properties of t. Usually it is not – e.g., for the edge colorings we get exactly the representation described in the example section below which is not optimal because boundaries which differ only by renaming colors always have the same coefficients.

But there are finite linear representations which cannot be modeled by real valued edge coloring models. For example counting colorings (both edge and vertex ones) if we treat them not as colorings but as partitions (i.e., the colors do not have identity):

Observation 5.41. *The number of k-partitions of vertices (resp. edges) which are a coloring cannot be calculated by a real valued edge coloring model for any $k > 1$.*

Proof. A graph with single vertex (resp. edge) has only one such coloring but disjoint union of two copies of this graph has 2 coloring – either the vertices (resp. edges) have the same color or not. Hence this parameter is not multiplicative and by Theorem 5.39 it cannot be calculated by a real valued edge coloring model. □

Also circuit double covers cannot be counted by a real valued edge coloring model:

Observation 5.42. *The number of circuit double covers is not edge reflection positive.*

Proof. We construct a quantum graph for which the number of CDCs is negative. Consider gadget g_1 of size 6 created from Petersen graph by removing two inner vertices not connected by an edge and gadget g_2 consisting of two disjoint cubic

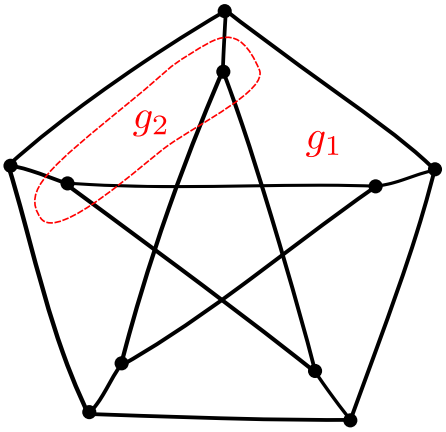

Figure 5.3: Petersen graph split into two 6-gadgets

vertices such that $\mathcal{J}_g(g_1, g_2) = \mathcal{J}_g(g_2, g_1)$ is Petersen graph (see Figure 5.3). Define quantum gadget $H = 3g_2 - g_1$. The number of CDCs of $\mathcal{J}_g(H, H)$ is less that -2 (computed by the program described in Section 6.3) so the number of CDCs is not edge reflection positive. □

Corollary 5.43. *Circuit double covers cannot be counted by a real valued edge coloring model.*

In general complex valued models are stronger than real valued ones. For example a real valued graph parameter $(-1)^{|E(G)|}$ cannot be represented by a real valued model but can be by a complex valued one as noted by Schrijver [2015]. So it is natural to ask whether we cannot use a complex model instead. The answer is negative:

Theorem 5.44. *Circuit double covers cannot be counted by a complexed valued edge coloring model.*

Proof. Theorem 5.40 shows that a linear representation derived from an edge coloring model has d^k boundaries. But we show below (Observation 6.3) that any linear representation of the number of circuit double covers has $2^{\Omega(k \log k)}$ boundaries. □

It is also worth noting that, despite the presented examples, an edge coloring model (and hence the derived linear representation) does not need to have a combinatorial interpretation. Szegedy [2007] showed that the value of the edge coloring model is invariant under action of an orthogonal group on a suitable polynomial ring. He also provided an example that the usual edge coloring model of the number of perfect matchings:

$$t(a, b) = \begin{cases} 1 & \text{if } a = 1, \\ 0 & \text{otherwise} \end{cases}$$

can be transformed into

$$t'(a, b) = 2^{-(a+b)/2}(a - b)$$

and this transformed model still calculates the number of perfect matchings.

5.6 Examples of Linear Representations

In this section we list a few examples of linear representations for common graph problems. We defer the linear representation of the number of CDCs to the next chapter (Chapter 6) as it is slightly more complicated and we also want to cover it in more detail. Note that we designed the framework to count objects. So although it might also be used for optimisation problems it usually leads to infinite representations.

For minimization problems it might be better to build analogous theory but with addition instead of multiplication and minimum instead of addition. In such a framework each coordinate would describe the best object with a given boundary. On the other hand it will no longer be a linear space so an extra care is needed. Because we are interested in counting, we do not pursue this direction any further.

5.6.1 Counting Vertices and Edges

The simplest example is probably counting the vertices of a graph. The most straightforward way is to define[8] $O^k = \mathbb{Q}$ for all k, let $h_{\mathcal{P}}$ map a gadget to the number of its vertices, let $\uplus_{\mathcal{P}}$ be addition and $f_{\mathcal{P}}$ and all other $\mathcal{J}_{\mathcal{P}}$ be identity. This is obviously a decomposable representation but not a linear one – we require linearity in all arguments $\mathcal{J}_{\mathcal{P}}(a_1x_2, a_2x_2) = a_1a_2\mathcal{J}_{\mathcal{P}}(x_2, x_2)$ which does not hold for addition. To fix this we change O^k to $\mathbb{Q} \times \mathbb{Q}$ and redefine

- $h_{\mathcal{P}}(g) = (|V(g)|, 1)$,
- $\uplus_{\mathcal{P}}((n_1, u_1), (n_2, u_2)) = (n_1u_2 + n_2u_1, u_1u_2)$, and
- $f_{\mathcal{P}}((n, u)) = n$.

The addition of the second coordinate which is always 1 may seem artificial at first but it is quite natural if you treat each coordinate as the number of partial objects of some type. In this case the objects we are interested in are (partial) vertices which might be modeled by partial functions $\{\emptyset\} \to V(g)$. So there are naturally two types of these objects on gadgets:

- Objects which selected a vertex in a given gadget (partial functions $\{\emptyset\} \to V(g)$ defined at $\emptyset$) – and there is always $|V(g)|$ of them.
- And objects which have not selected a vertex so it can be selected in other gadget (partial functions $\{\emptyset\} \to V(g)$ which are not defined at $\emptyset$) – and there is always exactly one object of this type.

The linear representation counting edges is exactly the same except $h_{\mathcal{P}}(g) := (|E(g)| - |g|/2, 1)$ (i.e., normal edges count for one but half-edges only for 1/2). These representations are optimal as can be seen using Observation 5.24 and two gadgets with different number of vertices (resp. edges) for each k.

[8] Note that formally we treat each O^k as independent copy of $\mathbb{Q}$ so they are disjoint from each other in all points except 0.

5.6.2 Colorings as Mappings

There are two natural ways to treat d-colorings (both edge and vertex ones): Either as functions into a set of colors $[d]$ or as partitions with at most d parts. For existential questions, both of these representations behave the same and mappings are usually a bit easier to work with. For counting, they differ – colorings as mappings which differ only by renaming colors are the same coloring when treated as partitions. We first look at colorings as mappings.

We define the representation counting edge d-colorings in the following way (we denote $\mathbf{1}_x$ the vector with 1 at position x and zeros elsewhere; we define the join functions on the basis of the linear space only):

- $R := \mathbb{Q}$, $B^k := [d]^k$ where $[d]^k$ is the set of all k-tuples of the numbers 1 to d denoting all the possible d-colorings of k half-edges,
- $(h_{\mathcal{P}}(g))_c$ for $c \in B^{|g|}$ is the number of edge d-colorings of g with colors c on the half-edges,
- $(\mathcal{J}_{i,j})_{\mathcal{P}}(\mathbf{1}_c) = \mathbf{1}_{c'}$ if $c_i = c_j$ and 0 otherwise where c' is c without the i-th and the j-th component,
- $\uplus_{\mathcal{P}}(\mathbf{1}_c, \mathbf{1}_{c'}) = \mathbf{1}_{c.c'}$ where . means concatenation,
- $\pi[\sigma]_{\mathcal{P}}(\mathbf{1}_c) = \mathbf{1}_{\sigma(c)}$, and
- $f_{\mathcal{P}}((n)) = n$ (note that $|B^0| = 1$).

Note that this is exactly the same representation we would get by taking the natural vertex model for the number of d-colorings and converting it into a linear representation by Theorem 5.40. Also this representation is not the optimal one because every gadget has the same coefficients for the boundaries which differ only by permuting the colors (e.g., $(1, 2, 1)$ and $(2, 1, 2)$). Using this observation we can keep only one boundary from each such group. This reduces the number of boundaries by the factor of almost $d!$.

The linear representation for vertex d-colorings is the same except for the following:

- we color every half-edge with the color of its only incident vertex,
- $(h_{\mathcal{P}}(g))_c$ for $c \in B^{|g|} = [d]^{|g|}$ is the number of vertex d-colorings of g with colors c on the half-edges, and
- $(\mathcal{J}_{i,j})_{\mathcal{P}}(\mathbf{1}_c) = \mathbf{1}_{c'}$ if $c_i \neq c_j$ and 0 otherwise where c' is c without the i-th and the j-th component.

Note that there exists a vertex model for vertex coloring due to Freedman et al. [2004] which proves the characterization of the parameters that can be counted by weighted homomorphisms (every vertex d-coloring is a homomorphism into K_d) and this characterization is the same as for the real-valued vertex models (which was proved by Szegedy [2007]).

5.6.3 Group Flows

Note that a general (M, B)-flow (Definition 3.1) can be represented just like an edge coloring with $|B|$ colors. The only difference is the multiplicity vectors assigned to the base gadgets.

5.6.4 Colorings as Partitions

Colorings as partitions are more complicated. Even the edge colorings cannot be represented by a real valued vertex model as shown in Observation 5.41. We will again use integers to denote the colors on the half-edges. Because the colors do not have identity now, we will consider descriptions which differ only by renaming colors to be the same – i.e., the boundary of size three $(1, 2, 1)$ is the same as boundary $(3, 1, 3)$. To remove ambiguity, we will always choose lexicographically minimal representation of the given boundary (e.g., $(1, 2, 1)$ in the previous example).

When joining gadgets together we have to try all the possible identifications between colors of each gadget. There is up to $d!$ such identifications but it might be less if one of the gadgets does not use all d colors. Given gadgets g_1 and g_2 each of them with a coloring using l_1 resp. l_2 colors, the possible identifications are in one-to-one correspondence with matchings M of complete bipartite graph $([l_1], [l_2], E)$ such that graph $([l_1], [l_2], M)$ has at most d components (i.e., edges denote identification of two colors and there is at most d colors in the end). This is the hard part – if all the colorings use d colors then[9] $c^d_m(g) = d!c^d_p(g)$ but if there is an unused color the relation is not that simple.

We denote $I_d(l_1, l_2)$ the set of all the identifications for colorings with l_1 and l_2 colors where the resulting coloring has at most d colors. For $i \in I_d(l, l')$ and colorings c, c' we denote $i(c, c')$ the coloring obtained by identifying colors of c and c' according to i and followed by concatenation, and $|i|$ the number of colors existing after the identification.

Note that knowing the colors on half-edges might not be enough – if some color is not used on the half-edges, we need to know whether it is used inside the gadget or not. So k-boundaries for vertex d-colorings will be

$$(c_1, c_2, \ldots, c_k \,|\, l\}$$

where $(c_1, \ldots, c_k)$ describes the colors on half-edges and among such colorings it is the lexicographically minimal one under renaming colors and l is the total number of colors used either on the half-edges or inside the gadget. Obviously the number of k-boundaries is at most d^{k+1}. We define the rest of the representation in the following way (all the linear functions are defined on the basis of the space):

- $R = \mathbb{Q}$, B^k are defined above,
- $(h_{\mathcal{P}}(g))_{(c_1,\ldots,c_k\,|\,l\}}$ for $(c_1, \ldots, c_k \,|\, l\} \in B^{|g|}$ is the number of edge d-colorings of g with colors $c_1, \ldots, c_k$ on the half-edges using l colors in total,
- $(\mathcal{J}_{i,j})_{\mathcal{P}}(\mathbf{1}_{(c\,|\,l\}}) = \mathbf{1}_{(c'\,|\,l\}}$ if $c_i = c_j$ and 0 otherwise where c' is c without i-th and j-th component,

[9] We denote the number of the edge colorings as mappings $c^d_m(g)$ and the number of the edge coloring as partitions $c^d_p(g)$ for a gadget g and d colors.

- $\uplus_{\mathcal{P}}(\mathbf{1}_{(c\,|\,l\}}, \mathbf{1}_{(c'\,|\,l'\}}) = \sum_{i \in I_d(l,l')} \mathbf{1}_{(i(c,c')\,||\,i|\}}$,
- $\pi[\sigma]_{\mathcal{P}}(\mathbf{1}_{(c\,|\,l\}}) = \mathbf{1}_{(\sigma'(c)\,|\,l\}}$ where $\sigma'(.)$ denotes permuting according to σ followed by choosing lexicographically minimal equivalent representation, and
- $f_{\mathcal{P}}(\mathbf{1}_{(\,|\,l\}}) = 1$.

To count the vertex colorings we can use the same trick as for the colorings-as-mappings.

6. Counting Double Covers II

In this chapter we continue the effort started in Chapter 4 and we show both an exponential lower bound for Flower snarks (Theorem 6.8) and an exponential lower bound for planar graphs (Theorem 6.14). We remind that the number of CDCs is denoted ν.

6.1 The Linear Representation

With the general framework in place, we can get back to the circuit double covers. We will model the number of CDCs as a finite linear representation over the field $\mathbb{R}$. We describe the model for cubic graphs and it might be possible to extend it to general graphs but we do not explore it further as cubic graphs are our main interest.

Because we describe CDCs by crossings on edges (see Section 4.2), it is natural to extend this definition to gadgets. The CDC on a gadget is a (multi)set of circuits and walks which covers every edge of the gadget twice (including the half-edges). Both ends of each walk must be half-edges and no edge or vertex can appear twice in one walk. The crossings on regular edges are already determined but the crossings on half-edges will be determined when the half-edge is joined with another half-edge creating a regular edge.

How do the gadget joins act on the CDCs? The disjoint union of the underlying gadgets is just a union of the CDCs and it always succeeds. Permuting half-edges does nothing, the only interesting operation is joining two half-edges together. When we are joining two half-edges, there are two walks on each of them and we must determine in which of the two possible ways can we join them.

Here comes the difference between cubic and general graphs: In cubic graphs constructed from $\mathcal{V}_3$ without free edges no two walks (or circuits) in a CDC can span the same set of edges so there are always two ways to do the join. But if we allow free edges or vertices of degree distinct from three, two walks in CDC might span exactly the same set of the edges. Then there is only one way to do the join because walks in a CDC do not have an identity on their own. As we are interested in cubic graphs, we will assume that no two walks in a CDC of a graph are identical.

What determines whether we can join two walks? We can always join a walk to itself, creating a circuit. If the walks are different, we only need them to not share an edge (otherwise we would create a self-touching walk which could not be completed into a circuit). Note that circuits do not participate in the joins in any way.

So to be able to join the two half-edges we need to remember for every walk which half-edges are its end points and which other walks it shares an edge with. We will record this in the following way: The walks will be numbered consecutively, starting at 1. For each half-edge we record the numbers of the two walks incident with it, e.g., the only possible configuration on boundary of size 3 can be written $((1,2),(1,3),(2,3)\,|\,\}$. Then for every two walks that share an edge and it is not yet known from the description of the half-edges, we record a tuple containing the numbers of these two walks, e.g., if walks 5 and 6 are incident

only inside the gadget, we would write $(\,|\,(6,5)\}$. If we were to allow CDCs which might include the same walk two times (i.e., non-cubic graphs) we also need to add which walks are the same otherwise we would overcount.

The same boundary can obviously written in many ways – the operations that preserve the same structure are renaming the walks, swapping the order of the numbers in each tuple, and permuting the tuples describing incidences inside the gadget. To get rid of this non-uniqueness we just take the equivalence classes under all this operations. In the implementation we represent each class by its lexicographically minimal element. In theoretical results, to simplify the notation, we use any element of the class to represent the whole class. So we might write

$$((1,2),(1,3),(2,3)\,|\,\} = ((1,2),(2,3),(3,1)\,|\,\}$$

instead of

$$[((1,2),(1,3),(2,3)\,|\,\}] = [((1,2),(2,3),(3,1)\,|\,\}]$$

as the boundaries are always the whole equivalence classes so there is no danger of confusion. Note that gadgets of size 0 – i.e., graphs – have only one boundary.

Multiplicity vector of a gadget (the value $h_\nu(g)$ for gadget g) describes how many CDCs with each boundary there are. The multiplicity vectors we get for gadgets are always non-negative in all coordinates. It should be obvious that such multiplicity vectors fully describe the gadgets in the terms of CDCs. The linearity follows from the formal definitions of the elementary joins below (Definition 6.1). We omit a formal proof that joins and h_ν commute, it would be trivial but even more technical than similar proof of Theorem 5.40.

As we observed above (Corollary 5.28), we want to allow free edges. We noted above that free edges break our assumption that two walks never span exactly the same set of edges. On the other hand we want to avoid extending our representation just because of them. Hence we just try to add them, modifying what the number of CDCs means for some edge cases if needed.

A free edge has two walks spanning the same half-edge. Hence there is only one option when joining to free edge but our representation would count two. To compensate for this we define a free edge to have the multiplicity vector $(1/2)$. It is easy to see that this works out well in all the cases except when a graph contains a vertex-less loop (note that a loop with a vertex can never have a CDC in a cubic graph). In this case the number of CDCs is divided by two for each vertex-less loop. We still call this parameter the number of circuit double covers even though it differs from ν as defined in Chapter 4 because they differ only for graphs with vertex-less loops and we care only about simple (cubic) graphs.

We denote ∂c the boundary of a CDC c and we denote $\gamma(g)$ the number of components of g which consist of free edges only. By applying a permutation σ on an n-tuple t we mean a tuple obtained by permuting the elements of t according to permutation σ. For tuples s and t, the expression $s;t$ denotes the concatenation of them such that the numbers (recursively) contained in t are shifted so they do not collide with numbers in s. The expression $t[w \to v]$ denotes the replacement of all occurrences of w in t by v. Finally $\xi_b(u \to v)$ is an indicator function whether $b[u \to v]$ is a correct boundary or not.

Definition 6.1. *We define a linear representation ν of the number of CDCs in the following way: The boundaries are the equivalence classes described above.*

The operations are (the notation is described above):

$$h_\nu(g) = (1/2)^{\gamma(g)} \sum_{c \text{ a CDC of } g} \mathbf{1}_{\partial c}$$

$$f_\nu(m) = m_{(\,|\,)}$$

$$\uplus_\nu^{k,k'}(m,m') = \sum_{(a\,|\,j\}\in B^k} \sum_{(a'\,|\,j'\}\in B^{k'}} \mathbf{1}_{(a;a'\,|\,j;j'\}} m_{(a\,|\,j\}} m'_{(a'\,|\,j'\}}$$

$$\pi^k[\sigma]_\nu(m) = \sum_{(a\,|\,j\}\in B^k} \mathbf{1}_{(\sigma(a)\,|\,j\}} m_{(a\,|\,j\}}$$

$$\xi_b(u\to w) = \begin{cases} \mathbf{1}_{b[u\to w]} & \text{if } b[u\to w] \text{ is a correct boundary of a CDC} \\ 0 & \text{otherwise} \end{cases}$$

$$(\mathcal{J}^k_{1,2})_\nu(m) = \sum_{\substack{b\in B^k \\ b=(((u_1,u_2),(w_1,w_2)).a\,|\,j\}}} (\xi_b(u_1\to w_1, u_2\to w_2) + \xi_b(u_1\to w_2, u_2\to w_1)) m_b$$

Note that due to the way h_ν is defined, the described linear parameter correctly counts the number of CDCs even if given CDC can be described by multiple boundaries (and we did not prove that there is a unique choice of the function ∂). In this case ∂ chooses an arbitrary boundary from those which describe the given CDC but a single CDC is still mapped to only one boundary so we do not overcount.

It is natural to ask whether this representation is the optimal one and how many boundaries of size k exist. We show that the described representation has $2^{\Theta(k^2)}$ boundaries of size k. We also show that any linear representation must have at least $2^{\Omega(k\log k)}$ boundaries so our representation is not that far from the optimal one. We do not expect this lower bound to be even asymptotically tight because the chosen gadgets are not the best possible. Below we show better bounds for small values of k. They prove that the described representation is not the best one but also they show that the optimal number of boundaries seems to be closer to the representation than the lower bound. But of course they are only a few datapoints so they might be the exceptional ones.

Observation 6.2. *The described linear representation has $2^{\Theta(k^2)}$ boundaries of size k.*

Proof. We show that the dominant part of the boundary description is whether the walks share an edge inside the gadget (the second part of the boundary description): Every two circuits might meet and there is $\binom{k}{2}$ of pairs of them. Up to k meets might be already forced by the part describing half-edges but it is still $\binom{k}{2} - O(k) = k^2/2 - O(k) = \Theta(k^2)$ binary choices.

The number of possible half-edge descriptions (the first part of the boundary description) is at most $\binom{k}{2}^k \le k^{2k} = 2^{2k\log k} = 2^{o(k^2)}$ but still at least 1. Together this gives $2^{\Theta(k^2)}\cdot 2^{o(k^2)} = 2^{O(k^2)}$ from above and $2^{\Omega(k^2)}\cdot 1$ from below. □

Observation 6.3. *Any linear representation counting the number of circuit double covers must have at least $2^{\Omega(k\log k)}$ boundaries of size k.*

Proof. We show that the matrix[1] $A^k_{g_1,g_2} = \nu(\mathcal{J}_g(g_1,g_2))$ has rank at least $2^{\Omega(k\log k)}$ which, combined with Observation 5.24, gives the lower bound.

[1] We remind that $\mathcal{J}_g$ is the gluing which joins the half-edges with the same labels.

Consider a matrix $C^k_{g_1,g_2}$ defined the same way as A^k but the columns and rows are not spanning all the k-gadgets but only a chosen subset of them S^k. Obviously C^k is a submatrix of A^k and $\text{rank}(C^k) \leq \text{rank}(A^k)$. The hard part is to choose the right S^k as we must be able to precisely calculate the number of CDCs of each graph created by joining elements of S^k.

We choose S^k to be the set of all so called diamond matchings. A diamond gadget is a 2-gadget created by cutting an edge of K_4. A diamond matching of size k is created by a disjoint union of $k/2$ diamond gadgets (so k must be even) and then permuting their half-edges. Now we show that there is $2^{\Omega(k \log k)}$ of diamond matchings of size k and that the corresponding matrix C^k has full rank.

We prove the first part by induction. There is one diamond matching of size 2. Now for even $k \geq 4$ we have $k-1$ possibilities where is the other half-edge of the diamond gadget who has the last half-edge. Now if we remove this gadget we have a diamond matching of size $k-2$. This leads to formula $(k-1)(k-3)(k-5)\ldots 1$ which is $2^{\Theta(k \log k)}$.

To show that C^k has full rank, it is enough to show $\det(C^k) \neq 0$. The key observation that $\nu(\mathcal{J}_g(g_1, g_2)) = 2^{2k-c}$ where c is the number of connected components of $\mathcal{J}_g(g_1, g_2)$. This is because we have two ways how to cover a diamond gadget and we have two ways how to join two diamond gadgets together but when we close the cycle we have only one possibility. The number of components is maximized when $g_1 = g_2$ – then there are $k/2$ components, each of them a necklace of length two. Whenever $g_1 \neq g_2$ the number of components is strictly smaller.

So the matrix C^k has some values 2^x on diagonal and all other values are also powers of two but strictly larger. Considering the definition of the determinant the sum contains element 2^{xl} for the diagonal (where l is the size of the matrix) and all other elements of the sum are multiples of 2^{xl+1}. Hence $\det(C^k) \equiv 2^{xl} \mod 2^{xl+1}$ so $\det(C^k) \neq 0$. □

On the other hand for small k the asymptotic behaviour is not important and the values itself are more interesting. We summarize our findings in Table 6.1. For k up to eight it contains the following:

- The number of k-boundaries of our representation ($|B^k|$). This determines the speed of our implementation.
- The rank of the matrix $\mathcal{J}^k_g$ which is an upper bound for the number of k-boundaries of the best representation $|B^k_{\mathcal{L}}|$ (due to Observation 5.26). This gives an upper bound on the speed of the hypothetical optimal implementation.
- The rank of matrix L^k which is the matrix constructed by Observation 5.24 of the highest rank we found and gives lower bound for $|B^k_{\mathcal{L}}|$. This gives a lower bound on the speed of any implementation of a linear representation counting CDCs.
- The number of diamond matchings of the given size – i.e., the lower bound proved by Observation 6.3. Note that diamond matchings are defined only for even k. This shows the best lower bound for $|B^k_{\mathcal{L}}|$ we currently know without using computer.

Table 6.1: Number of CDC boundaries of given size

k	0	1	2	3	4	5	6	7	8
$\lvert B^k \rvert$	1	0	1	1	33	744	69,920	13,710,912	?
$\text{rank}(\mathcal{J}_g^k)$	1	0	1	1	21	202	?	?	?
$\text{rank}(L^k)$	–	–	–	–	21	161	?	?	?
$\lvert S^k \rvert$	1	–	1	–	3	–	15	–	105

For trivial reasons our representation is optimal for $k \leq 3$. The question marks in the table stand for what we do not know because the computation took too long to finish. The set of gadgets used to construct matrix L^4 are cyclic ladder gadgets of size 2 up to 7 with all the permutations of all the half-edges except one. This gives 36 gadgets and matrix B of rank 21. For details see experiment `cdc_4_boundaries.py`. For L^5 we used cyclic ladders with one half-edge joined to a cubic vertex, again of sizes 2 up to 7. Increasing size of the ladders further did not help. For details see experiment `cdc_5_boundaries.py`.

As a toy example to get used to working with this framework we will determine the exact number of outer-fixed CDCs of a flower (Definition 4.6, do not confuse with Flower snarks). We show that the lower-bound in Observation 4.7 is quite tight. Note that this example slightly deviates from the description shown above because we are covering two edges of the boundary only once not twice. On the other hand it demonstrates that it is easy to modify the framework and it is also one of a few examples small enough to be done by hand (although we used a computer to find the formula first, see experiment `flowers.py`).

Theorem 6.4. *The number of outer-fixed circuit double covers F_k of flower of size k is*

$$F_k = \frac{2^{k-1} + (-1)^k}{3} + 1.$$

Proof. The flower gadget is a gadget obtained from flower by cutting it along a ray coming out of its central face (see an example in Figure 6.3). Flower gadget of size k is obtained by joining k step gadgets (Figure 6.2) in the obvious manner. We choose the flower gadget of size 2 as the initial gadget.

The only boundaries with non-zero coefficients for any flower gadget of size at least 2 are:

1. $((1),(1,2),(2,3),(3)\,|\,\}$,
2. $((1),(2,3),(1,2),(3)\,|\,(1,3)\}$ and
3. $((1),(2,3),(2,3),(1)\,|\,(1,2),(1,3)\}$.

Those boundaries were determined by a computer but they can be also found by hand by starting with the base gadget, repeatedly joining the step gadget and stopping when the set of the boundaries stabilizes. The multiplicity vector of the step gadget is:

Boundary	Coefficient
$((1),(1,2),(2,3),(3)\,\lvert\,(1,3)\}$	1
$((1),(2,3),(1,2),(3)\,\lvert\,(1,3)\}$	1

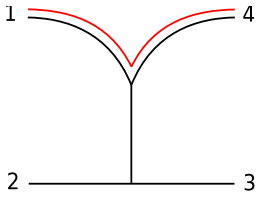

Figure 6.2: The step gadget of a flower

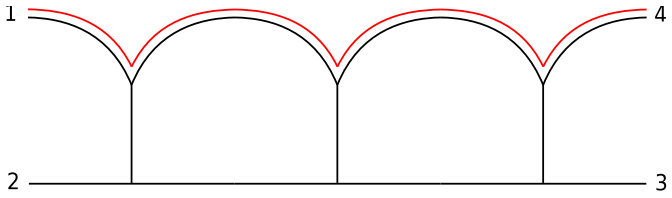

Figure 6.3: The flower gadget of size 3

Combining all the boundaries of a flower gadget and the support of the multiplicity vector of the step gadget, we obtain the step matrix and the following formula for the number of CDCs of a k-flower:

$$F_k = (1,0,1)\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 1 & 0 \end{pmatrix}^{k-2}\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

We transform the matrix into its Jordan normal form

$$F_k = (1/2,1,-1)\begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix}^{k-2}\begin{pmatrix} 4/3 \\ 1 \\ -1/3 \end{pmatrix}$$

and obtain the formula (using Corollary 5.36):

$$F_k = \frac{2^{k-1}+(-1)^k}{3}+1. \qquad \square$$

Another simple exercise is to show that replacing a cubic vertex with a triangle doubles the number of CDCs.

Observation 6.5. *Replacing a 3-vertex with a triangle doubles number of circuit double covers.*

Proof. There is only one boundary of size 3, vertex of degree 3 has multiplicity vector (1) and triangle has multiplicity vector (2). (See Figures 4.1 and 4.2.) $\square$

Although a very simple observation, it gives us an infinite class of graphs with exactly $2^{n/2-1}$ CDCs.

Corollary 6.6. *An n-vertex graph created from three parallel edges by repeatedly expanding vertices to triangles has exactly $2^{n/2-1}$ circuit double covers.*

Using our implementation of the described linear representation we calculated the number of circuit double covers of the three smallest snarks.

Lemma 6.7. *Petersen graph has 52 circuit double covers, first Blanusa snark 6966 and second Blanusa snark 6389.*

Proof. Calculated by a computer using the framework described above. See the `small-snarks.py` experiment. $\square$

Table 6.4: The eigenvector for the eigenvalue 16 of the $\nu(J_k)$

Boundary	Coefficient
$((1,2),(1,2),(3,4),(3,4),(5,6),(5,6)\,\vert\,a\}$	1
$((1,2),(1,2),(3,4),(3,5),(5,6),(4,6)\,\vert\,a\}$	4
$((1,2),(1,2),(3,4),(5,6),(5,6),(3,4)\,\vert\,a\}$	1
$((1,2),(1,3),(3,4),(2,4),(5,6),(5,6)\,\vert\,a\}$	4
$((1,2),(1,3),(3,4),(2,5),(5,6),(4,6)\,\vert\,a\}$	8
$((1,2),(1,3),(3,4),(4,5),(5,6),(2,6)\,\vert\,a\}$	8
$((1,2),(1,3),(3,4),(5,6),(5,6),(2,4)\,\vert\,a\}$	4
$((1,2),(1,3),(4,5),(2,4),(3,6),(5,6)\,\vert\,a\}$	8
$((1,2),(1,3),(4,5),(2,6),(3,6),(4,5)\,\vert\,a\}$	4
$((1,2),(1,3),(4,5),(4,5),(3,6),(2,6)\,\vert\,a\}$	4
$((1,2),(1,3),(4,5),(4,6),(3,6),(2,5)\,\vert\,a\}$	8
$((1,2),(3,4),(3,4),(1,2),(5,6),(5,6)\,\vert\,a\}$	1
$((1,2),(3,4),(3,4),(1,5),(5,6),(2,6)\,\vert\,a\}$	4
$((1,2),(3,4),(3,4),(5,6),(5,6),(1,2)\,\vert\,a\}$	1
$((1,2),(3,4),(3,5),(1,2),(4,6),(5,6)\,\vert\,a\}$	4
$((1,2),(3,4),(3,5),(1,5),(4,6),(2,6)\,\vert\,a\}$	8
$((1,2),(3,4),(3,5),(1,6),(4,6),(2,5)\,\vert\,a\}$	8
$((1,2),(3,4),(3,5),(5,6),(4,6),(1,2)\,\vert\,a\}$	4
$((1,2),(3,4),(5,6),(1,2),(3,4),(5,6)\,\vert\,a\}$	1
$((1,2),(3,4),(5,6),(1,5),(3,4),(2,6)\,\vert\,a\}$	4
$((1,2),(3,4),(5,6),(5,6),(3,4),(1,2)\,\vert\,a\}$	1
Where $a = ((i,j) : i,j \in [6], i < j)$ are all the possible touches so every walk touches all the other walks.	

We also calclulated the asymptotic number of CDCs of Flower snarks. The calculations suggest that the right constant should be around $1/720$. We can prove (utilizing a computer comuptation) that $c \geq 1/(720 + 10^{-100})$ but we omit the proof.

Theorem 6.8. *Flower snark*[2] J_k *has* $c16^k \pm O(15^k)$ *circuit double covers for some constant* $c > 0$.

Proof. We look at Flower snarks as a graph sequence. Hence we can write the numbers of CDCs they have as

$$\nu(J_k) = m_{\text{fin}} A^{k-2} m_{\text{init}}$$

for suitable vectors m_{init} and m_{fin} and a matrix A. The matrix A is too large (10148×10148) to obtain its Jordan normal form. Instead we calculated that its largest eigenvalue is 16, found its eigenvector, verified that 16 has algebraic multiplicity one and all other eigenvalues have absolute value strictly smaller than 15 (for details see below and also experiment `flower-snarks.py`).

Using this knowledge we can see that the Jordan normal form of A will have a block of size one corresponding to the eigenvalue 16 and all the other blocks

[2]To be precise, only J_k with odd $k \geq 3$ are snarks (some definitions exclude even J_3 as it contains a triangle), J_k with even k are edge 3-colorable graphs. Anyway we calculate the number of CDCs for both odd and even k.

correspond to eigenvalues with absolute value less than 15. Applying the Corollary 5.36 we obtain the bound because $x^k\texttt{poly}(k) \in O(15^k)$ for all $|x| < 15$.

Description of the experiment: We calculated the vectors m_{fin} and m_{init} and the matrix A. We guessed the largest eigenvalue using the ratios of $\nu(J_{k+1})/\nu(J_k)$ which seem to converge to 16 quickly. We verified that 16 is an eigenvalue by checking that nullity (i.e., size minus rank) of the matrix $B = A - 16E$ (where E is the identity matrix) is not zero. The nullity – which is also the geometric multiplicity of the given eigenvalue – of B is one.

To find the eigenvector we considered the ratios $r_i = (A^8 m_{\text{init}})_i/(A^7 m_{\text{init}})_i$ and selected the set I of all the coordinates i for which this ratio was at least 15 (and in particular it existed). We took $s = \min_{i\in I}(A^8 m_{\text{init}})_i$ and calculated the vector

$$u'_i = \begin{cases} \frac{(A^8 m_{\text{init}})_i}{s} & \text{if } i \in I, \\ 0 & \text{otherwise.} \end{cases}$$

The resulting vector u is obtained by rounding each coordinate of u' to the nearest integer and it shown in Table 6.4. We verified that it is an eigenvector and we also checked that there exists a coordinate which is non-zero in both u and m_{fin}.

Then we calculated matrix C – a deflation of A using the vector u. The deflation changes the eigenvalue corresponding to the used eigenvector to zero but keeps all the other eigenvalues (but it might change their eigenvectors). The deflation algorithm used is due to Wielandt [1944] and it calculates

$$C = A - \frac{1}{u_p} u A_{p,*}$$

where $A_{p,*}$ is the p-th row of A and p is such an integer that $u_p \neq 0$. We bound the 1-norm

$$\|X\|_1 = \max_c \sum_r |X_{r,c}|$$

of the matrix C by $\|C^{32}\|_1 < 15^{32}$. Hence the spectral radius $\rho(C)$ (the largest absolute value of its eigenvalues) is less then 15 because $\rho(C)^k \leq \|C^k\|$ for every matrix norm $\|.\|$ (for details see, e.g., Section 5.6 of Horn and Johnson [2012]). This proves that the algebraic multiplicity of the eigenvalue 16 of A is one and that all the other eigenvalues of A are in the absolute value less then 15. □

We also calculated exact formulas for the number of CDCs of cyclic ladders (also called prisms) and crossed cyclic ladders:

Theorem 6.9. *Cyclic ladder of length k has L_k circuit double covers and crossed cyclic ladder has L^c_k circuit double covers (for $k \geq 3$) where:*

$$L_k = \frac{4^{k-1} + 9 \cdot 2^k - (-2)^{k-1} - 15k + 3(-1)^k(k-1) - 9}{6}$$
$$L^c_k = \frac{4^{k-1} + 9 \cdot 2^k - (-2)^{k-1} - 15k - 3(-1)^k k}{6}$$

Proof. Calculated by a computer using the framework described above. See the `ladders.py` experiment. □

The results we obtained so far motivate our following conjecture:

Conjecture 6.10. *Every bridgeless cubic graphs with n vertices has at least $2^{n/2-1}$ circuit double covers.*

Note that Corollary 6.6 shows that this conjecture cannot be any stronger as there is an infinite family of graphs for which this conjecture is tight. On the other hand a stronger wersion might hold for triangle-free or more cyclically connected graphs. In our search of all $\{C_3, C_4\}$-free cubic biconnected graphs on 20 or less vertices, the one closest to the bound is Petersen graph with ratio 3.25. Two more tested graphs had the ratio $\nu(G)/2^{|V(G)|/2-1}$ less than 10, all other had higher ratios. For details see Figure 6.5. The blue points represent tested graphs and the purple line is the lower bound of Conjecture 6.10. The data were obtained by experiment `test_exp_cdc.sh`.

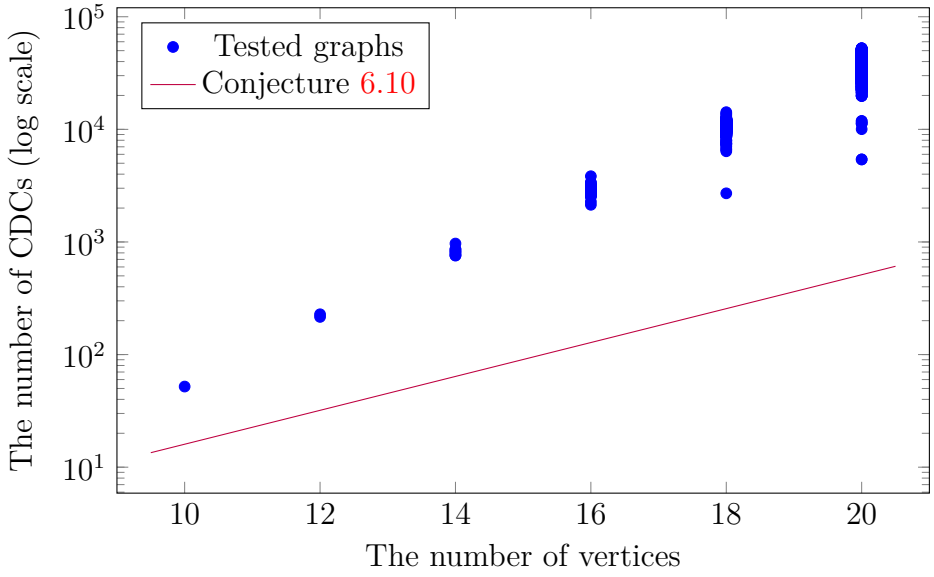

Figure 6.5: The number of CDCs of $\{C_3, C_4\}$-free cubic biconnected graphs

6.2 Reducing Cycles

In this section we combine the framework with linear programming to obtain a better bound on the number of CDCs of planar graphs. First we describe the method in general and then we apply it to the number of CDCs of planar graphs. It is also straightforward to use this method for other linear representations.

6.2.1 General Method

We want to show that graphs in some class $\mathcal{C}$ have many CDCs and we know that there is a small set of gadgets $\mathcal{S}$ such that every graph in $\mathcal{C}$ either has some gadget of $\mathcal{S}$ as an induced subgraph or it is trivial in some sense. We denote the class of the trivial cases $\mathcal{B} \subset \mathcal{C}$. The usual reasons for a graph to be considered trivial are a small number of vertices and existence of small cuts.

We can for every gadget $s \in \mathcal{S}$ choose a set of gadgets with fewer vertices $\mathcal{R}_s$ and try to prove that the number of CDCs of a graph G containing s can be bounded from below by the number of CDCs of G with s replaced by elements of

$\mathcal{R}_s$ and that these graphs also belong to $\mathcal{C}$. This allows us to proceed by induction on the number of vertices.

A bit more formally (for an application see the proof of Theorem 6.14): Suppose we are proving a lower bound of a form $c^{n(G)-d}$ where $c > 1$, $d \in \mathbb{R}$. Then we want to show the inequality

$$\nu(\mathcal{J}_g(g,s)) \geq \min_{r \in \mathcal{R}_s} c^{n(s)-n(r)} \nu(\mathcal{J}_g(g,r))$$

where $n(g)$ is the number of vertices[3] of a gadget g. Suppose this is true for all $\mathcal{J}_g(g,s) \in \mathcal{C}$, all the graphs $\mathcal{J}_g(g,r)$ also belong to $\mathcal{C}$ and we have other means to prove the bound for graphs in $\mathcal{B}$. Then we can prove the desired lower bound $c^{n(G)-d}$ for every $G \in \mathcal{C}$ by induction on the number of vertices using this formula as the induction step for the non-trivial graphs.

Proving this formula for each s is where the linear programming comes into play. We saw a special case of this approach before in Observation 6.5. But in that case there was only one boundary and one substitution gadget, so no linear program was needed. The idea of the program is to fix the value of the right-hand side and search for as small left-hand side as possible. Because we cannot search all gadgets, we search all possible multiplicity vectors instead.

Theorem 6.11. *Let $c > 1$, s, $\mathcal{R}_s$ and $n(.)$ be defined as above and to simplify the notation[4] put $\mathcal{J} = \mathcal{J}_g$. If the objective value of the linear program P (described below) is at least one then the following holds for all gadgets g:*

$$\nu(\mathcal{J}(g,s)) \geq \min_{r \in \mathcal{R}_s} c^{n(s)-n(r)} \nu(\mathcal{J}(g,r))$$

where $n(g)$ is the number of vertices of gadget g and the linear program P is:

$$\begin{aligned} \min_{m \in \mathbb{R}^{B^{|s|}}} \quad & \mathcal{J}_\nu(m, h_\nu(s)) \\ \forall r \in \mathcal{R}_s : 1 \leq \quad & c^{n(s)-n(r)} \mathcal{J}_\nu(m, h_\nu(r)) \\ 0 \leq \quad & m \end{aligned}$$

Proof. We remind that all the images of joins $\mathcal{J}_\nu$ are linear in each of their arguments and that $h_\nu(g)$ for a fixed gadget g is a constant vector. We show the more general inequality for every multiplicity vector m:

$$\mathcal{J}_\nu(m, h_\nu(s)) \geq \min_{r \in \mathcal{R}_s} c^{n(s)-n(r)} \mathcal{J}_\nu(m, h_\nu(r)).$$

Fix any $\alpha > 0$ and consider the linear program:

$$\begin{aligned} \min_{m \in \mathbb{R}^{B^{|s|}}} \quad & \mathcal{J}_\nu(m, h_\nu(s)) \\ \forall r \in \mathcal{R}_s : \alpha \leq \quad & c^{n(s)-n(r)} \mathcal{J}_\nu(m, h_\nu(r)) \\ 0 \leq \quad & m \end{aligned}$$

[3] Do not confuse with $|g|$ which denotes the size of the gadget, i.e., the number of its half-edges.

[4] This theorem holds with an unchanged proof for any other binary join but $\mathcal{J}_g$ is the only join we need due to Observation 5.4.

Note that for the optimal solution of this linear program the condition

$$\forall r \in \mathcal{R}_s : \alpha \leq c^{n(s)-n(r)} \mathcal{J}_\nu(m, h_\nu(r))$$

is tight (i.e., its right-hand side is α) for at least one r. (Otherwise $m = 0$ but then also $\mathcal{J}_\nu(m, .) = 0$.) If its objective value is at least α, then the desired inequality holds. But the choice of α does not matter because we can scale m accordingly. Hence we choose $\alpha = 1$ obtaining the linear program P. □

Note that if the objective value of the linear program is less than one but still more than zero then exponential bound for a smaller c might hold. This theorem also holds for any other linear representation which maps gadgets to non-negative vectors.

The downside of a linear program is that it is usually solved by a numerical method which is not suitable for a theoretical proof. We circumvent this by solving the dual problem. Note that any solution of the dual gives us a lower bound but of course suboptimal solutions will give weaker bounds. So for a given solution of the dual we only need to certify that it is indeed a solution which is easy and we do not need to prove optimality.

6.2.2 Application to Planar Graphs

We are interested in bridgeless cubic planar graphs. We know that every such graph contains a cycle of size at most five because its dual is also a planar graph and so it contains a vertex of degree at most five (due to Euler's formula). Moreover, for $c \leq \sqrt{2}$ and $d \leq 2$ we may reduce the cuts of size two and three due to Observations 4.8 and 4.9. So we take all bridgeless planar cubic graphs as the class $\mathcal{C}$ and we define $\mathcal{B}$ to be all graphs in $\mathcal{C}$ which are not cyclically 4-edge-connected.

We need to be able to replace 4-cycles and 5-cycles. The important observation is that if the graph is 3-edge-connected then all the 4-cycles and 5-cycles are faces. What can we replace them with? We need the replacements to be smaller. Ignoring the labeling of the half-edges we have the following options: one tree and one free edge for the 4-cycle and one possible tree and a combination of a cubic vertex and a free edge for the 5-cycle.

We tested all of them and the trees were never required to get the best results. Hence we will replace the 4-cycles with the two possible non-crossing choices of two free edges and the 5-cycles with a cubic vertex and free edge (again drawn in a non-crossing way) in all the 5 possible rotations. The following theorems show the results of this replacements:

Theorem 6.12. *Let G be a cyclically 4-edge-connected cubic graph with a 4-cycle. Let G_1 and G_2 be the two possible graphs obtained from G by deleting two opposite edges of the 4-cycle and suppressing vertices of degree 2. Then $\nu(G) \geq 4 \min\{\nu(G_1), \nu(G_2)\}$.*

Proof. Because the graph is cyclically 4-edge-connected and we are deleting non-adjacent edges, the resulting graph is still 2-edge-connected due to Observation 1.3. We apply Theorem 6.11 with $c = \sqrt{2}$, 4-cycle as s and the two non-crossing choices of two free edges $\mathcal{R}_s$. We obtain the following linear program:

$$\max \sum_{i=1}^{33} o_i m_i$$

$$1 \leq \sqrt{2}^4 \sum_{i=1}^{33} a_i m_i$$
$$1 \leq \sqrt{2}^4 \sum_{i=1}^{33} b_i m_i$$

where m_i are the variables and o_i, a_i and b_i are constants computed by experiment `reduce-cycle.py` (although they might be computed by hand in this case). Each variable corresponds to a boundary and $|B^4| = 33$ hence there is 33 variables. Each inequality corresponds to an elements of $\mathcal{R}_s$. Plugging in the values, taking dual and removing conditions obviously implied by other conditions, we get:

$$\begin{aligned} \max \frac{1}{4}x_0 + \frac{1}{4}x_1 \\ 2 \geq x_0 \\ 2 \geq x_1 \end{aligned}$$

The objective value of this linear program is 1. This satisfies the conditions of the theorem so we obtain:

$$\nu(G) = \nu(\mathcal{J}(g,s)) \geq \min_{r \in \mathcal{R}_s} \sqrt{2}^4 \nu(\mathcal{J}(g,r)) = 4 \min\{\nu(G_1), \nu(G_2)\}. \qquad \square$$

Theorem 6.13. *Let G be a cyclically 4-edge-connected cubic graph with a 5-cycle and no 4-cycle. Let $G_1, G_2, \ldots, G_5$ be the 5 possible graphs obtained from G by replacing the 5-cycle by a cubic vertex and an edge in non-crossing way (assuming the 5-cycle is a face). Then $\nu(G) \geq 5/2 \min_i \nu(G_i)$. If we replace the 5-cycle by a cubic vertex and an edge in all possible ways (i.e., breaking planarity) we get $\nu(G) \geq 3.75 \min_i \nu(G_i)$.*

Proof. We can simulate this replacement by removal of the two edges adjacent with the 5-cycle and the two vertices which should be connected by an edge in the result, contracting 5-cycle into a vertex and adding an edge to join the two vertices of degree two. These two vertices are distinct otherwise there would be a 4-cycle in the graph. So the deleted edges were not adjacent and the resulting graph is 2-edge-connected due to Observation 1.3.

The rest of the proof is analogous to the proof of the previous theorem and the computer aided part is also a part of experiment `reduce-cycle.py`. The non-crossing case with $c = \sqrt[4]{5/2}$:

$$\begin{aligned} \max\ & 0.4x_0 + 0.4x_1 + 0.4x_2 + 0.4x_3 + 0.4x_4 \\ \forall i : 1 &\geq x_i \\ 1 &\geq x_2 + x_4 \\ 1 &\geq x_1 + x_4 \\ 1 &\geq x_1 + x_3 \\ 1 &\geq x_0 + x_3 \\ 1 &\geq x_0 + x_2 \end{aligned}$$

The solutions is $x_i = 0.5$ and the objective value is 1. The crossing case with $c = \sqrt[4]{3.75}$ and all the 10 possible replacement gadgets (5 non-crossing plus 5 crossing) leads to a larger linear program so we omit it. But again the objective value is one, so we can apply the Theorem 6.11. $\square$

In both theorems the modified graphs have four vertices less than the original ones so the best we can prove is that the number of CDCs increases by the factor 2.5 (due to 5-cycles) with addition of four vertices. So we obtain:

Theorem 6.14. *Every bridgeless cubic planar graph has at least* $(5/2)^{n/4-1/2}$ *circuit double covers.*

Proof. By induction on n, the number of vertices of the graph G. The base cases are three parallel edges and K_4 because these are the only planar cubic graphs without a non-trivial cut. The three parallel edges graph has only one CDC which exactly matches the bound. The K_4 has two CDCs and the bound requires only approximately 1.58.

Suppose $n \geq 6$. If G has non-trivial cut of size two, we apply Observation 4.8 and we obtain

$$\begin{aligned}\nu(G) &\geq 2\nu(G_1)\nu(G_2) \geq 2(5/2)^{|V(G_1)|/4-1/2}(5/2)^{|V(G_2)|/4-1/2} \\ &= 2(5/2)^{n/4-1/2}\sqrt{2/5} \geq (5/2)^{n/4-1/2}\end{aligned}$$

which we needed. Similarly for a non-trivial 3-cut we use Observation 4.9:

$$\begin{aligned}\nu(G) &\geq \nu(G_1)\nu(G_2) \geq (5/2)^{|V(G_1)|/4-1/2}(5/2)^{|V(G_2)|/4-1/2} \\ &= (5/2)^{(n+2)/4-1} = (5/2)^{n/4-1/2}.\end{aligned}$$

So G is cyclically 4-edge-connected. It is 3-edge-connected so each facial walk in its planar embedding is a circuit. Due to Euler's formula, the planar dual of G must have a vertex of degree at most five. Hence G has a face of size at most five. We already excluded 3-faces because the cut around a triangle is a non-trivial cut of size three. If G has a 4-face, we apply Theorem 6.12:

$$\nu(G) \geq 4(5/2)^{(n-4)/4-1/2} = \frac{4 \cdot 2}{5}(5/2)^{n/4-1/2} \geq (5/2)^{n/4-1/2}.$$

Otherwise G has a 5-face and we apply Theorem 6.13:

$$\nu(G) \geq (5/2)(5/2)^{(n-4)/4-1/2} = (5/2)^{n/4-1/2}.$$ □

To compare this with Conjecture 6.10, $(5/2)^{n/4} = 2^{cn}$ for c approximately 0.33 so this is still a weaker bound than Conjecture 6.10 asks for. We conclude this section with a summary of what we know about hypothetical counterexample to Conjecture 6.10:

Corollary 6.15. *A minimal counterexample (the one with the smallest number of vertices) to Conjecture 6.10:*

1. *does not have 2-edge-cut,*
2. *does not have non-trivial 3-edge-cut,*
3. *does not contain triangle,*
4. *does not contain 4-cycle, and*
5. *has at least 22 vertices.*

The first three points are due to Observations 4.8 and 4.9, the fourth one due to Theorem 6.12 and the last one was verified by a computation on all 3-edge-connected cubic graphs up to 20 vertices (see `test_exp_cdc.sh` experiment). Note that we cannot exclude 5-cycles as the bound provided by the second part of Theorem 6.13 is too weak.

6.3 Implementation

In this section we discuss the implementation of the proposed framework. We used this implementation to obtain all the computer aided results in this chapter. Readers interested only in the theoretical results may skip this section. We start by some general notes then we describe the core of the implementation and we conclude this section with some notes on implemented representations and graph sequences. For more details consult the implementation which is available at our department GitLab:

`https://gitlab.kam.mff.cuni.cz/radek/cdc-counting`.

6.3.1 General Notes

The main difference of the implementation from the theoretical approach is that the multiplicity vectors are represented sparsely as lists of named tuples and only indices with non-zero value are present. Also the joins are not explicit matrices but Python functions.

The advantages of this approach are mainly that the program consumes much less memory as usual multiplicity vectors contain many zeros and the matrices of the join functions even more. Another advantage is that this representation does not need to enumerate all the boundaries of given size which might not be easy for some representations and it is impossible for linear representations which are not finite. Note that to be able to do the computations we do not need the representation to be finite but only that the support of every multiplicity vector we get is finite which is much weaker condition. This representation also allows implementation of some representations which are not linear.

The implementation is written in Python 3 [van Rossum and Drake, 2009] – it was tested with CPython 3.8[5] – and uses Sage [The Sage Developers, 2021] whenever standard functionality like graphs or matrices are needed. We chose Python mainly because it is a high level language with no direct access to pointers which by itself prevents a lot of annoying bugs related to memory management when compared with C [Kernighan and Ritchie, 1988] or C++ [Stroustrup, 2013]. Another reasons for choosing Python are built-in long integers and Sage which contains (among other) graph and linear algebra algorithms.

6.3.2 Base Data Types

File `base.py` contains the base data types on which the rest of the library is built on. The high-level structure is following: Every graph parameter (or more precisely its decomposable representation) is modeled by an object of a class

[5] `https://github.com/python/cpython/tree/3.8`

deriving from `GraphParameterBase`. Gadgets are modeled as objects of classes deriving from `Gadget`. The most important methods for the users are listed below; `p` is an object of class `GraphParameterBase` corresponding to representation $\mathcal{P}$ and `g` is an object of class `Gadget` corresponding to gadget g.

- `Gadget.join` which joins gadgets (described below).
- `g.eval_gadget(p)` which corresponds to $h_{\mathcal{P}}(g)$.
- `p.finalize` which corresponds to $f_{\mathcal{P}}$.
- `g.eval(p)` which is a shortcut for `p.finalize(g.eval_gadget(p))` (i.e., $\mathcal{P}(g)$ – it calculates the value of the parameter for gadget g given g is a graph).

A short descriptions of the defined types follows (for a more detailed exposition consult the source code):

`BoundaryValue` is a named tuple `(boundary, value, origins)` representing a (non-zero) coefficient of a multiplicity vector. Member `boundary` contains the boundary in the sense of linear representations (see Definition 5.13, i.e., (the representation of) some element of $B_{\mathcal{P}}$) and `value` is its coefficient. Member `origins` allows tracking how the value was created during gadget joins. It can be used to enumerate the objects which we are counting.

`Gadget` is an abstract class representing a gadget. It has three subclasses: `BaseGadget` representing base gadgets (currently only a cubic vertex and a free edge) and `JoinGadget` for gadgets created by joining other gadgets – these are enough to calculate the value of the parameters on any graph. The last subclass – `FakeGadget` – is used when a creation of specific multiplicity vector is needed (although it might not correspond to any gadget) – most notably when constructing matrix A from Observation 5.34.

The cubic vertex and the free-edge are defined as constants `CUBIC_VERTEX` and `FREE_EDGE`. The most important method of `Gadget` is a static method

```
join(gadgets, joins, outs)
```

for joining gadgets together. Its parameters are a list of gadgets to join, description which half-edges should be joined together and a list of the half-edges of the resulting gadget to allow permuting them easily. The parameter `joins` is a list where each its element is a tuple of half-edges which will be joined together.

The half-edges are described as tuples `(gadget_index, edge_index)` where both indices start at one. Each half-edge must appear exactly once in parameter list. For example a triangle can be created the following way (see also Figure 6.6):

```
Gadget.join([ CUBIC_VERTEX ]*3,
  [ ((1, 2), (2, 1)), ((2, 2), (3, 1)), ((3, 2), (1, 1)) ],
  [ (1, 3), (2, 3), (3, 3) ])
```

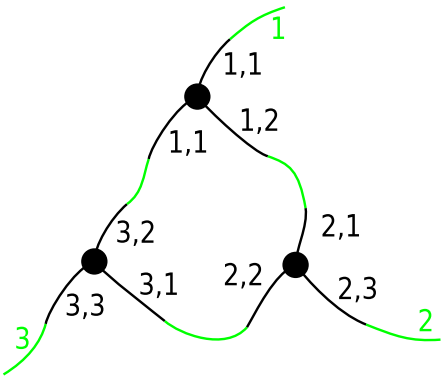

Figure 6.6: An example of `Gadget.join`. Black part are the original gadgets. Green parts are the new connections.

`FakeGadget` is a subclass of `Gadget`. It does not represent any real gadget but instead is used to inject given multiplicity vector into the computation. Its constructor takes two parameters, the size of the gadget it should represent and the desired multiplicity vector (as list of `BoundaryValue`).

`GraphParameterBase` is the abstract base class for all the graph parameter representations. All objects of this class and its subclasses are required to be immutable and hashable. Its main method of the interest is

```
eval_join(self, multiplicity_vectors, joins, outs, offsets)
```

which evaluates the parameter on the join of given `multiplicity_vectors`. It is called by the implementation of `JoinGadget.eval_gadget`. Its implementation is sketched in Algorithm 6.7. It starts with a Cartesian product of the non-zero coordinates of all input multiplicity vectors, applying `join_boundaries` to each tuple (note that it returns list of the `BoundaryValues`). Then it applies all the required edge joins and continues with the canonization of the generated boundaries. At the end it collects all the values with the same boundary together.

Every subclass of `GraphParameterBase` must have an instance attribute `CUBIC_VERTEX` and optionally also `FREE_EDGE` with the multiplicity vectors of these base gadgets and it must implement the following methods (the `self` parameter is omitted):

- `join_boundaries(tuple_of_boundaries, lengths)`: Join given tuple of boundaries into one boundary. The parameter `lengths` gives the sizes of the boundaries as a speed optimization.
- `join_edges(boundary, edge1, edge2)`: Join given half-edges in the given boundary.
- `project_and_canonize(selector, boundary)`: Transform given boundary into the canonical form keeping only half-edges present in `selector` in the given order.
- `finalize(multiplicity_vector)`: Calculate the value $f_{\mathcal{P}}$ of the given multiplicity vector.

Input: `self`, list of multiplicity vectors of the subgadgets M, list of the half-edges to join J, list of output half-edges O
Output: Multiplicity vector of the described join gadget

$l \leftarrow [\,]$
for $p \in$ *cartesian_product*(M) **do**
    $l \leftarrow l +$ `self.join_boundaries(p)`
end for
for $(e_1, e_2) \in J$ **do**
    $l' \leftarrow [\,]$
    for $b \in l$ **do**
        $l' \leftarrow l' +$ `self.join_edges(`b`,` e_1`,` e_2`)`
    end for
    $l \leftarrow l'$
end for
`pc` $\leftarrow$ `self.project_and_canonize`
$l \leftarrow [$`BoundaryValue(pc(`O`,` b`.boundary),` b`.value)` $: b \in l\,]$
$B \leftarrow \{b$`.boundary` $: b \in l\}$
return $[$*BoundaryValue*$(b,\ \sum_{b' \in l,\, b'.\mathit{boundary}=b} b'.\mathit{value})$ $: b \in B\,]$

Algorithm 6.7: Method `GraphParameterBase.eval_join`

`SimpleParameterBase` As a convenience we implement class `SimpleParameterBase` which can be used to implement linear parameters which can be described by coloring edges. It requires only implementation of method

```
is_compatible(boundary, edge1, edge2)
```

and the attribute `CUBIC_VERTEX`. This method should return `True` if given half-edges can be joined together. For example parameter `EdgeThreeColoring` might be implemented by setting `CUBIC_VERTEX` to all six 3-coloring of edges around it and defining:

```
def is_compatible(self, b, e1, e2):  return b[e1] == b[e2].
```

`GraphSequenceBase` is the abstract base class of all the graph sequences described in Definition 5.33. It is not exported from the module but instead it is utilized through the class decorators `ParametrizedGraphSequence` and `GraphSequence`. The following properties must be defined for each sequence either as class properties or instance properties. They are listed with the corresponding parts of Definition 5.33.

- `sequence_start` – z, the index offset of the sequence.
- `base_gadget` – g_{init}, the initial gadget.
- `step_gadget` – g_{step}, the step gadget.

- `step_join` and `step_out` – $\mathcal{J}_{\text{step}}$, the description of the step join. The next gadget in the sequence is obtained by

  ```
  Gadget.join([ g, step_gadget ], step_join, step_out)
  ```

  where `g` is the previous gadget of the sequence.

- `final_join` – $\mathcal{J}_{\text{fin}}$, the final join as the second parameter to `Gadget.join`. Note that there is not `final_out` because the result should be a graph.

The important methods of `GraphSequenceBase` are:

- `gadget(k)` which returns k-th gadget.

- `graph(k)` which returns k-th graph G_k.

- `stabilize(parameter)` which explicitly calculates vectors u, v and a matrix M such that $\mathcal{P}(G_k) = uM^{k-z}v$ for the parameter $\mathcal{P}$ represented by `parameter`. It can also obtain a formula for this value using function `matrix_to_formula` described below. It is called `stabilize` because it does not enumerate all boundaries of the given size but it instead computes the step gadgets until their support stabilizes.

6.3.3 Parameters

We have implemented the following graph parameters: `VertexCount`, `EdgeColoring`, `GroupFlow`, `VertexColoring`, `UnderlyingGraph`, `CycleDoubleCover` and `CircuitDoubleCover`. Parameter `CircuitDoubleCover` implements the linear representation described in Section 6.1. `UnderlyingGraph` constructs given graph as an object of Sage's `Graph` class. This is useful because `Gadget` is internally a tree of joins with base gadgets as leaves.

`EdgeColoring` is implemented using `SimpleParameterBase` by defining the value of a cubic vertex and method `is_compatible` to be `b[e1] == b[e2]`. The representation of `VertexCount` is defined similarly by `b[e1] != b[e2]` and the representation of `GroupFlow` by `b[e1] == -b[e2]`. `CycleDoubleCover` is treated as a special case of group flows over $\mathbb{Z}_2^k$ for suitable k.

6.3.4 Graph Sequences

The implemented graph sequences are:

- `Necklace` – A necklace gadget is obtained by cutting an edge of K_4 and a necklace of size k is obtained by joining k necklace gadgets into a cycle.

- `CyclicLadder` also called a prism. This sequence is parametrized by a boolean value determining whether the result is crossed or not. A non-crossed cyclic ladder of size k is obtained by taking two C_k and joining their corresponding vertices by edges. See Figure 5.2. A crossed cyclic ladder is obtained from non-crossed one by cutting the last edges of both cycles and joining the two cycles together.

- `GeneralizedPetersen` The generalized Petersen graph with parameters s, k is constructed the following way: Take C_k, append a vertex to every vertex of this cycle. We call the new vertices outer. Now join i-th outer vertex with $(i + s)$-th outer vertex (calculating modulo k). This gives a cubic graph.

  The sequence is parametrized by s. So the usual Petersen graph can be obtained by `GeneralizedPetersen(2).graph(5)`. The graphs with $s = 1$ are non-crossed cyclic ladders.

- Two implementations of Flower snarks (Definition 1.11) – `FlowerSnark` and `FlowerSnarkAlt`. `FlowerSnark` implements the construction from the definition which crosses the outer edges only once in the base gadget. `FlowerSnarkAlt` was implemented to double check our results and it construct Flower snarks by crossing edges at every gadget. The constructions create isomorphic graphs for odd k but different ones for even k.

- `Flower` implements gadgets from the flower construction (Definition 4.6). It uses `FakeGadget` to cover the outer edges only once. Hence it works only with the `CircuitDoubleCover` parameter and it depends on its implementation.

6.3.5 Miscellaneous & Utils

Below we note a few of miscellaneous and utility functions which are either generally useful or implement some interesting algorithms.

- `graph_to_gadget` converts a Sage `Graph` into a gadget so parameters can be evaluated on it. It takes an optional parameter describing how the graph should be decomposed into gadgets. If none is given it tries to guess some which does not create too large cuts in the process. Although we try to be a bit smart we do not give any guarantees about the quality of the guessed decomposition. An interesting option would be to use, e.g., path-width decomposition to construct the decomposition.

- `edge_model_join` implements gluing $\mathcal{J}_g$ – the join which joins the half-edges with the same labels.

- `parameter_matrix` given a parameter $\mathcal{P}$ and a list of k-gadgets $g_1, \ldots, g_n$, it calculates an $n \times n$ matrix A such that $A_{i,j} = \mathcal{P}(\mathcal{J}_g(g_i, g_j))$. Note that the rank of A gives a lower bound on the number of k-boundaries of any representation of the same parameter as shown in Observation 5.24.

- `enumerate_diamond_matchings` enumerates all diamond matchings of given size. The diamond matchings are used in the proof of Observation 6.3 although the proof does not use computer.

- `matrix_to_formula(u, M, v)` implements Corollary 5.36 using Sage's symbolic calculation facilities.

7. Voltage Graphs

This chapter is a report on a work-in-progress which was started during Robert Šámal's stay at Simon Fraser University (SFU), Canada in 2019/20. It is a join work with Matt DeVos and Bojan Mohar from SFU. Although it is a kind of generalization of Berman et al. [2017], it was discovered independently. Berman et al. [2017] rephrased the snark construction of Loupekine into the language of voltage graphs and then used the list of known small snarks to construct a few new infinite families of snarks with interesting properties.

Their general approach was to select a suitable snark, turn it into a voltage graph over $\mathbb{Z}_k$ and then to prove some properties about graphs sequence obtained by increasing k. Our approach is similar but instead of starting with small snarks, we start with voltage graphs. We conducted extensive search on small template graphs and small groups, tested their derived graphs and if the derived graph was interesting, examined the whole sequence.

For now we only processed some of the voltage graphs over $\mathbb{Z}_k$ and we created the graph sequences by increasing k. We use the framework built in the previous chapters to examine whether a sequence contains infinitely many snarks. The computational complexity of this approach strongly depends on sum of the absolute values of the assignment (interpreted as integers). Hence there still remains a lot of sequences we did not test yet. On the other hand a big advantage of our method is that when the computation finishes we know exactly which graphs in the sequence are snarks and which are not, there are no open cases left. In the future we also want to test graphs obtained from a non-cyclic or even non-abelian groups.

In the next section we describe the basics of voltage graphs, then we describe the used construction of a graph sequence from a single voltage graph and, very briefly, the implementation of the computer programs we used. We conclude this chapter with an interesting graph sequence we found.

7.1 Definitions and Properties

Voltage graphs are an elegant way for describing large graphs using smaller ones. We start with a formal definition. Note that even though we allow the group to be non-abelian, we use additive notation because we will mostly work with groups $\mathbb{Z}_k$. Also similarly to group flows we need an oriented graph but the orientation does not matter because when we flip an orientation of an edge we can just take inverse of its label and obtain the same derived graph.

Definition 7.1 (Voltage Graph)**.** *Let Γ be a group. A directed graph $G = (V, E)$ (we allow loops and parallel edges) together with a function $\varphi : E \to \Gamma$ is a* voltage graph*. We call G the* template *and φ the* assignment*. We define $\Gamma(\varphi) = \Gamma$.*

If the group is $\mathbb{Z}_k$, we can also label edges with integers and interpret them modulo k. This representation gives us ability to change k and hence obtain a graph sequence. The derived graph is a graph on vertices $V(G) \times \Gamma$ where edges are determined by labels on edges of G. Because voltage graphs might contain parallel edges and loops, we need to be a little careful with the notation. We use

u_e and v_e to denote the head and tail of an edge e (to match the usual notation $uv \in E$).

Definition 7.2 (Derived Graph)**.** *Let (G, φ) be a voltage graph. Its* derived graph*, denoted $D(G, \varphi)$, is an undirected graph with vertex set $V(G) \times \Gamma(\varphi)$ and edges*

$$\left\{\{(u_e, \gamma), (v_e, \gamma + \varphi(e))\} : e \in E(G), \gamma \in \Gamma(\varphi)\right\}.$$

A derived graph might contain loops or parallel edges but we do not care about such cases. Hence we assume that a derived graph is simple and loopless. As we already noted, the orientation of the voltage graph does not matter. Moreover, derived graphs are also invariant under more complex operations on the voltage graphs. For abelian groups this operation is best imagined as (a series of) adding a fixed value ρ to all edges in some edge cut which just shifts one side of the cut in the derived graph by ρ.

Observation 7.3. *Let (G, φ) be a voltage graph and let $\rho : V(G) \to \Gamma(\varphi)$ be any function. Define $\varphi'(e) = \rho(u_e) + \varphi(e) - \rho(v_e)$. Then $D(G, \varphi) \cong D(G, \varphi')$.*

Proof. We prove that mapping m defined by $m((u, \gamma)) = (u, \gamma - \rho(u))$ is an isomorphism from $D(G, \varphi)$ to $D(G, \varphi')$. Edges of $D(G, \varphi)$ are

$$\left\{\{(u_e, \gamma), (v_e, \gamma + \varphi(e))\} : e \in E(G), \gamma \in \Gamma(\varphi)\right\},$$

m maps them to

$$\left\{\{(u_e, \gamma - \rho(u_e)), (v_e, \gamma + \varphi(e) - \rho(v_e))\} : e \in E(G), \gamma \in \Gamma(\varphi)\right\}$$

and by shifting γ from right by $\rho(u_e)$ we get

$$\left\{\{(u_e, \gamma), (v_e, \gamma + \rho(u_e) + \varphi(e) - \rho(v_e))\} : e \in E(G), \gamma + \rho(u_e) \in \Gamma(\varphi)\right\}$$

which are exactly the edges of $D(G, \varphi')$. □

A straightforward application of this is to fix a spanning tree in the template graph and assume that all its edges are labeled zero (the identity in Γ). This will greatly speed up the search through all the possible assignments.

Corollary 7.4. *Let G be a template and T its spanning tree. If D is a derived graph of (G, φ) for some φ then there exists $\varphi' : E(G) \to \Gamma(\varphi)$ such that $D \cong D(G, \varphi')$ and φ' is zero on all edges of T.*

7.2 The Program

We give a high-level overview of the program used to find the interesting snarks. We omit the implementation details as the code is not published yet. We will make it publicly available when we consider it mature enough and publish some results. It will be available at our gitlab

https://gitlab.kam.mff.cuni.cz/radek/voltage-graphs.

We are mainly interested in the cubic graphs. Hence although the core of our program can work with general simple graphs, certain parts – most notably some filters (described below) – work only with cubic graphs. As noted above, the core of our program is an enumeration of all possible voltage graphs and their derived graphs given a fixed template graph and a group.

The implementation of the enumeration is recursive: It assigns a value to the given edge e and recursively calls itself on the next edge without a value. After returning from the recursion, it assigns another value to e, recurses again and so on until all the possible values of e are enumerated. The complete high-level description is in Algorithm 7.1.

Input: Template graph T, group Γ, mapping $s : E(T) \to 2^\Gamma$, `invariants` and `filters`
Output: Derived graphs

```
1  expand_edge(0)
2  function expand_edge(e) begin
3      if e ≥ |E(T)| then
4          if G has loops or parallel edges then return
5          for f ∈ filters do
6              if not f.eval(G) then return
7          end for
8          print G
9          return
10     end if
11     for v ∈ s(e) do
12         Construct edges in the derived graph G corresponding to e labeled
           with v overwriting the ones previously created.
13         for i ∈ invariants do
14             if not i.eval(G) then continue
15         end for
16         expand_edge(e + 1)
17     end for
18     Remove the edges we constructed.
19 end function
```

Algorithm 7.1: The enumeration of the derived graphs

This approach has two main weaknesses: It generates a lot of graphs which are not interesting for some trivial reason (they might not be connected, contain loops or parallel edge, etc.) and it generates many graphs which are isomorphic. We provide a few ways to overcome these problems. They differ in expressive power, computational cost and the phase of the graph generation in which they are applied.

The simplest tool is the restriction of edges in the template to only some members of the group Γ. We call this restriction *value sets*. Corollary 7.4 shows that we can set edges of a spanning tree to zero and not lose any derived graph but dramatically decrease the number of generated graphs (i.e., the number of

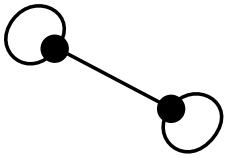

Figure 7.2: Two loops connected by an edge

repetitions). Value sets are applied during the enumeration of the edge and as such provides a great speed up by pruning the whole subtrees of the tree of the recursion. But value sets are applied to every edge separately and it might be beneficial to restrict the values of an edge depending on other edges.

For example consider the graph shown in Figure 7.2. We fix the value zero on the non-loop edge. But we also want to prevent generating the graphs which differ only by swapping the values on the loops. We call the tool to help here *invariants*. Invariants are equations about values assigned to the edges which must be true. Currently we support two types of invariants.

First one asserts that the sum of some edges must be zero or non-zero (we allow specifying that edge should be reversed before summing). The second one that the representation of the value on one edge is either strictly or non-strictly smaller than the representation of the value on the other edge. The group elements are represented by integers from 0 to $|\Gamma| - 1$. The implementation does not promise anything about the mapping from the group elements to integers except that the identity element of the group is represented by the integer zero. However, the cyclic groups are mapped in the natural way by identity.

Hence to fix our example we add the invariant $l_1 \leq_r l_2$ where l_1 and l_2 are the values of the two loops and $\leq_r$ is the implementation defined order on the group elements. The invariant is tested immediately after all its edges are assigned values. The last tool is filters.

Filters

Filters are applied after the whole graph is generated and they are given both the assignment and the derived graph. They are also able to store a state during processing of one template. There are currently the following filters:

- `Sort:` Sort neighbours of every vertex and check if we have already seen this graph. It removes obviously isomorphic graphs and it is very fast.
- `Connected:` Check whether the graph is connected. Note that bridges are not an issue as usually, depending on the assignment, the derived graph is either 3-connected or not connected at all.
- `Traces:` Use Traces[1] [McKay and Piperno, 2014] to transform the graphs into a canonical form under (a subset of) their automorphism group and remove duplicities. It takes parameter `orbits` which limits the considered (potential) automorphisms. It weakens the filter but might significantly speed up its evaluation. Sometimes it might be beneficial to use this filter several times with larger and larger orbits as a speed optimization.

[1] Of nauty and Traces, https://pallini.di.uniroma1.it/.

- `CyclicConnectivity`: Remove graphs with girth (the length of the shortest circuit), local girth or cyclic connectivity (Definition 1.2) lower than given thresholds. The girth and local girth calculation uses breadth first search run from each vertex until a circuit is found. We color vertices by the edges of the starting vertex to avoid reporting lollipops (a path connected to a circuit) as circuits which would mess up local girth calculation.

  Local girth is $\max_{v \in V} \min_{C \ni v} |C|$, i.e., for every vertex v we consider the shortest circuit passing through v and we take the maximal length of those over all vertices. A large difference between girth and local girth hints that the graph is far from being highly symmetric.

  We use the quadratic algorithm ($\mathcal{O}(n^2 \log^2 n)$ to be precise) of Dvořák et al. [2004] to calculate cyclic connectivity. The paper states that this algorithm works for graphs on at least 243 vertices. The inspection of the algorithm shows that it is enough to find k disjoint trees such that each of them is adjacent to at least $k+1$ edges where k is the cyclic connectivity we want to test (hence $\mathcal{O}(\log n)$ by Dvořák et al. [2004]) and 243 vertices assures this. We instead try to always apply the quadratic algorithm and if we fail to find the disjoint trees (which we select greedily), we stop and use their $\mathcal{O}(n^3 \log n)$ algorithm instead. Both algorithms and hence also this filter work only on cubic graphs.

- `CanonicalAssignment`: This filter attempts to remove isomorphic graphs from output but unlike Sort and Traces it does not work with the derived graph but with the assignment. The high-level idea is that we want to use automorphisms A of the template graph to remove assignments leading to isomorphic derived graphs. This filter is obviously weaker than Traces but it might be faster (and hence it can used to do partial filtering before using Traces).

  We already use the normalization of the assignment due to Corollary 7.4. We denote $N(.)$ the function which does this normalization assuming we have some fixed spanning-tree.[2] Permuting the edge labels of an assignment a by $\sigma \in A$ leads to an isomorphic voltage graph and thus to an isomorphic derived graph. However, $\sigma(a)$ is most likely non-normalized yielding no improvement. But the assignment $N(\sigma(a))$ might be generated so removing it might help.

  We define $C_a = \{N(\sigma(a)) : \sigma \in A\}$. Let P be the set of assignments which we passed for further processing and let x be a newly generated assignment. We keep the set $S = \bigcup_{a \in P} C_a$. We discard x if $x \in S$, otherwise we pass x for further processing and we add x into P. To reduce the memory consumption, we map all possible normalized assignments to integers and store S as a bit set.

  We would like to think about C_a as an equivalence class but note that this might not be the case, i.e., there might exist an assignment a and permutations $\sigma, \pi \in A$ such that $N(\pi(N(\sigma(a)))) \notin C_a$. If C_a are not

[2] This is a bit of simplification. The actual implementation also normalizes values on loops (if we replace x with $-x$ on a loop, the derived graph stays the same) and some fixed non-tree edge.

equivalence classes, the filter is weaker than it might be but it is still correct – it removes only isomorphic graphs. Also if C_a are not equivalence classes, it might help to add C_x to S even if we discarded x. But we currently do not do this as our intuition is that the gain is not worth the extra time consumed.

7.3 A New Family of Snarks

As noted before, we used the described program on small templates with all possible assignments in several selected small cyclic groups. When we found an interesting snark (mainly cyclically 5-connected) we transformed its voltage graph into a graph sequence by converting the assignment from $\mathbb{Z}_k$ by interpreting it as integers in range $(-k/2, k/2]$ and then increasing k. The choice of $(-k/2, k/2]$ is arbitrary from the theoretical point of view. Other choices might lead to different sequences which may or may not contain snarks. We use this range because it minimizes the size of the sequence and thus the running time of our algorithm – the size of the sequence is $2\sum_i |a_i|$ where a is the assignment vector. Then we tested whether elements of this sequence are at least cyclically 4- or 5-connected and which elements are snarks – i.e., not 4-edge-colorable. This is easy to check by hand because all we need to check is one element of the sequence:

Observation 7.5. *Let G_n be the derived graph of some fixed voltage graph over $\mathbb{Z}_n$. Let $m = \max_i |a_i|$ where a_i are the assigned values as integers. If the girth of G_n is k and $mk < n$ then the girth of G_j is k for all $j \geq n$. Similarly if the cyclic connectivity of G_n is k and $mk < n$ then the cyclic connectivity of G_j is k for all $j \geq n$.*

Proof. Obvious. Both the shortest cycle and the smallest nontrivial cut might either go around the whole cycle of gadgets (see Figure 7.3) or not. If $mk < n$, then k edges are too few for the cycle to go around. We prove the claim for the cyclic connectivity by contradiction.

For contradiction let $j > n$ be the smallest integer such that G_j has cyclic connectivity $k' < k$. Let $C \subset E(G_j)$ be a non-trivial cut of the size k' in G_j. Then by pigeonhole principle there exists i such that if we remove the i-th step gadget[3] (obtaining a graph isomorphic to G_{j-1}) we do not remove any edge belonging to C. Hence C is a non-trivial cut in G_{j-1}. Contradiction. □

To check whether a graph is a snark, we count its 3-edge-colorings using the representation described in Section 5.6.2 – its boundaries are all possible 3-colorings of the half-edges. We start calculating the multiplicity vectors m_i of the gadgets in the sequence g_i and continue doing so until we find two multiplicity vectors with the same support. This will happen as there are only 3^s possible supports (where $s = 2\sum_i |a_i|$). Note that the support is all we care about as this representation always maps non-negative vectors to non-negative vectors. Hence if $\mathrm{supp}(m_i) = \mathrm{supp}(m_j)$ for $i < j$ then $\mathrm{supp}(m_{i+l}) = \mathrm{supp}(m_{i+(l \bmod (j-i))})$ for all $l > 0$.

[3] We consider edges joining i-th gadget with $(i+1)$-th up to $(i+m)$-th gadgets to belong to the i-th gadget.

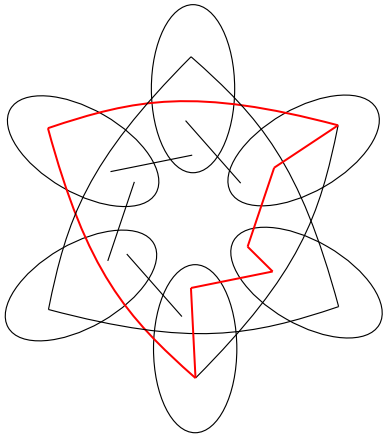

Figure 7.3: An example of a cycle "going around" in a graph G_6

We started testing the small sequences first as the running time is exponential in the size of the sequence. Hence a lot of sequences we found were covered by results of Berman et al. [2017]. The first one which is obviously not covered by them (because it has three non-loop edges with non-zero labels) is shown in Figure 7.4. We summarize its properties in the following observation. We do not include a bound on the number of CDCs because the graph sequence was too large to be processed by our hardware.

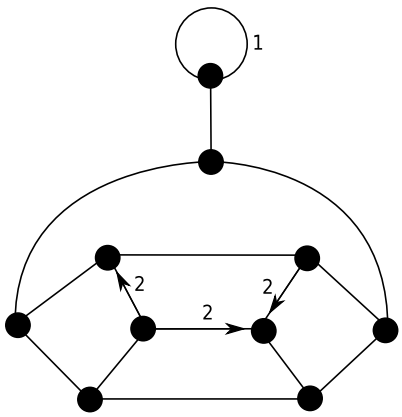

Figure 7.4: Voltage graphs without 4-cycles. The unlabeled edges have labels 0.

Theorem 7.6. *Let G_i be the derived graph of the voltage graph in Figure 7.4 over $\mathbb{Z}_i$ for $i \geq 3$. Then the girth of G_i is $\min\{i, 5\}$ and G_i is cyclically 5-connected with exceptions of $i = 8$ which is cyclically 4-connected, $i = 3$ or 6 which are cyclically 3-connected and $i = 2$ which is cyclically 2-connected. Moreover G_i is not 3-edge-colorable (and hence it is a snark) if and only if $i \bmod 4 \neq 0$.*

Proof. The girth and the cyclical connectivity were calculated for G_3 up to G_{11} which is enough by Observation 7.5. For 3-edge-colorability we calculated the supports of the multiplicity vectors and discovered that $\operatorname{supp} G_7 = \operatorname{supp} G_3$ and among $G_3, \ldots, G_6$ only G_4 has 3-edge-coloring. For technical details see experiment `nice-voltage.py`. □

Conclusion

We conclude this thesis with a summary of the open questions. Our main question from Chapter 2 was already positively resolved by Han et al. [2020]. This completed the last piece about group connectivity for 3-connected graphs. On the other hand the same question for 2-connected graphs is still open:

Conjecture 2.17. *Let Γ_1 and Γ_2 be abelian groups. Then there exists a graph which is Γ_1-connected but not Γ_2-connected.*

In Chapter 3 we increased the importance of the original conjecture of Matt DeVos (Conjecture 3.4) by observing that it implies existence of cycle double covers with a small number of cycles. We also conjecture that our strengthening holds for every graph and the smallest group in which it has nowhere-zero flow:

Conjecture 3.12. *For every graph G the strong homomorphism property holds for group $\mathbb{Z}_k$ where k is minimal such that G admits a nowhere-zero $\mathbb{Z}_k$-flow.*

The main open question about finite linear representations in general is characterization of the parameters for which they exist:

Problem 5.32. *Characterize graph parameters which have a finite linear representation over $\mathbb{Q}$, $\mathbb{R}$ or $\mathbb{C}$. What if we restrict growth of $|B^k|$?*

In the Chapter 6 we present a conjecture which is a natural strengthening of the Cycle Double Cover conjecture to counting:

Conjecture 6.10. *Every bridgeless cubic graphs with n vertices has at least $2^{n/2} - 1$ circuit double covers.*

We also show that if our version holds, it is tight for infinitely many graphs. On the other hand we do not know any graph for which the bound is tight and the graph does not contain a triangle. So there might be a room for improvement if we restrict ourselves to triangle-free graphs (or $\{C_3, C_4\}$-free or if we require higher cyclic connectivity).

Bibliography

Georgij M. Adelson-Velskij and A. Titov. On 4-chromatic cubic graphs. *Vopr. Kibernet*, 1974. in Russian.

Omid Amini, Louis Esperet, and Jan Van Den Heuvel. A unified approach to distance-two colouring of graphs on surfaces. *Combinatorica*, 33(3):253–296, 2013. doi: 10.1007/s00493-013-2573-2.

Kenneth I. Appel and Wolfgang Haken. Every planar map is four colorable. Part I: Discharging. *Illinois Journal of Mathematics*, 21(3):429 – 490, 1977. doi: 10.1215/ijm/1256049011.

Kenneth I. Appel, Wolfgang Haken, and John A. Koch. Every planar map is four colorable. Part II: Reducibility. *Illinois Journal of Mathematics*, 21(3):491 – 567, 1977. doi: 10.1215/ijm/1256049012.

Arash Asadi, Zdeněk Dvořák, Luke Postle, and Robin Thomas. Sub-exponentially many 3-colorings of triangle-free planar graphs. *J. Combin. Theory Ser. B*, 103 (6):706–712, 2013. doi: 10.1016/j.jctb.2013.09.001.

H. W. Becker and John Riordan. The arithmetic of bell and stirling numbers. *American Journal of Mathematics*, 70(2):385–394, 1948. doi: 10.2307/2372336.

Stefan Behnel, Robert Bradshaw, Craig Citro, Lisandro Dalcin, Dag Sverre Seljebotn, and Kurt Smith. Cython: The best of both worlds. *Computing in Science & Engineering*, 13(2):31–39, 2011.

Leah W. Berman, Déborah Oliveros, and Gordon I. Williams. Cyclic pseudo-Loupekine snarks, 2017. https://arxiv.org/abs/1707.05294.

Jean C. Bermond, Bill Jackson, and François Jaeger. Shortest coverings of graphs with cycles. *Journal of Combinatorial Theory, Series B*, 35(3):297–308, 1983. doi: 10.1016/0095-8956(83)90056-4.

Danilo Blanuša. Problem četiriju boja. *Glasnik Mat. Fiz. Astr.*, 1946.

Gunnar Brinkmann, Kris Coolsaet, Jan Goedgebeur, and Hadrien Mélot. House of graphs: a database of interesting graphs. *Discrete Applied Mathematics*, 2013. doi: 10.1016/j.dam.2012.07.018. URL http://hog.grinvin.org.

Harold S. M. Coxeter. My graph. *Proceedings of the London Mathematical Society*, s3-46(1):117–136, 1983. doi: 10.1112/plms/s3-46.1.117.

Matthew J. DeVos. *Flows on graphs*. Princeton University, 2000. PhD thesis.

Matthew J. DeVos. A homomorphism problem for flows. *Open Problem Garden*, 2007. http://www.openproblemgarden.org/op/a_homomorphism_problem_for_flows, [retrieved 2020-02-28].

Reinhard Diestel. *Graph theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, Berlin, fifth edition, 2017. doi: 10.1007/978-3-662-53622-3.

Jan Draisma, Dion C. Gijswijt, László Lovász, Guus Regts, and Alexander Schrijver. Characterizing partition functions of the vertex model. *Journal of Algebra*, 350(1):197 – 206, 2012. doi: 10.1016/j.jalgebra.2011.10.030.

Zdeněk Dvořák, Jan Kára, Daniel Král', and Ondřej Pangrác. An algorithm for cyclic edge connectivity of cubic graphs. In Torben Hagerup and Jyrki Katajainen, editors, *Algorithm Theory - SWAT 2004*, pages 236–247, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

Zdeněk Dvořák, Bojan Mohar, and Robert Šámal. Exponentially many nowhere-zero $\mathbb{Z}_3$-, $\mathbb{Z}_4$-, and $\mathbb{Z}_6$-flows. *arXiv e-prints*, art. arXiv:1708.09579, Aug 2017.

Joanna A. Ellis-Monaghan and Iain Moffatt. *Graphs on Surfaces: Dualities, Polynomials, and Knots*. Springer New York, 2013.

Paul Erdős, Arthur L. Rubin, and Herbert Taylor. Choosability in graphs. In *Proceedings of the West Coast Conference on Combinatorics, Graph Theory and Computing (Humboldt State Univ., Arcata, Calif., 1979)*, Congress. Numer., XXVI, pages 125–157. Utilitas Math., Winnipeg, Man., 1980.

Louis Esperet, František Kardoš, Andrew D. King, Daniel Král', and Serguei Norine. Exponentially many perfect matchings in cubic graphs. *Adv. Math.*, 227(4):1646–1664, 2011. doi: 10.1016/j.aim.2011.03.015.

Genghua Fan. Integer flows and cycle covers. *Journal of Combinatorial Theory, Series B*, 54(1):113–122, 1992. doi: 10.1016/0095-8956(92)90069-A.

Herbert Fleischner. Spanning eularian subgraphs, the splitting lemma, and petersen's theorem. *Discrete Mathematics*, 101(1):33–37, 1992. doi: 10.1016/0012-365X(92)90587-6.

Michael Freedman, Lovász László, and Alexander Schrijver. Reflection positivity, rank connectivity, and homomorphism of graphs. *Journal of the American Mathematical Society*, 20, 05 2004. doi: 10.1090/S0894-0347-06-00529-7.

Delbert R. Fulkerson. Blocking and anti-blocking pairs of polyhedra. *Mathematical Programming*, 1971. doi: 10.1007/BF01584085.

Martin Gardner. Mathematical games. *Scientific American*, 234(4):126–130, 1976.

Luis A. Goddyn, Michael Tarsi, and Cun-Quan Zhang. On (k, d)-colorings and fractional nowhere-zero flows. *J. Graph Theory*, 28(3):155–161, 1998. doi: 10.1002/(SICI)1097-0118(199807)28:3<155::AID-JGT5>3.0.CO;2-J.

Andrew Goodall, Thomas Krajewski, Guus Regts, and Lluís Vena. A tutte polynomial for maps. *Combinatorics, Probability and Computing*, 27(6):913–945, 2018. doi: 10.1017/S0963548318000081.

George Grätzer. *Universal Algebra*. 2008. doi: 10.1007/978-0-387-77487-9.

Herbert Grötzsch. Zur Theorie der diskreten Gebilde, VII: Ein Dreifarbensatz für dreikreisfreie Netze auf der Kugel. *Wiss. Z. Martin-Luther-U., Halle-Wittenberg, Math.-Nat. Reihe*, 1959.

G. Haggard. Edmonds characterization of disc embedding. In *Proceeding of the 8th Southeastern Conference of Combinatorics, Graph Theory and Computing*, pages 291–302, Winnipeg, 1977.

Miaomiao Han, Jiaao Li, Xueliang Li, and Meiling Wang. Group connectivity under 3-edge-connectivity. *Journal of Graph Theory*, 96, 09 2020. doi: 10.1002/jgt.22623.

Pavol Hell and Jaroslav Nešetřil. The core of a graph. *Discrete Mathematics*, 109 (1):117–126, 1992. doi: 10.1016/0012-365X(92)90282-K.

Ian Holyer. The NP-completeness of edge-coloring. *SIAM Journal on Computing*, 10(4):718–720, 1981. doi: 10.1137/0210055.

Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, USA, 2nd edition, 2012.

Radek Hušek, Peter Korcsok, and Robert Šámal. On girth of minimal counterexample to 5-flow conjecture. In *Bordeaux Graph Workshop*, 2016.

Radek Hušek, Lucie Mohelníková, and Robert Šámal. Group connectivity: $\mathbb{Z}_4$ vs $\mathbb{Z}_2^2$. *Journal of Graph Theory*, 93, 08 2019. doi: 10.1002/jgt.22488.

Radek Hušek and Robert Šámal. Homomorphisms of cayley graphs and cycle double covers. *Electronic Notes in Discrete Mathematics*, 61:639 – 645, 2017. doi: 10.1016/j.endm.2017.07.018. The European Conference on Combinatorics, Graph Theory and Applications (Eurocomb'17).

Radek Hušek and Robert Šámal. Homomorphisms of cayley graphs and cycle double covers. *The Electronic Journal of Combinatorics*, 27, 04 2020. doi: 10.37236/8456.

Radek Hušek and Robert Šámal. Counting circuit double covers. In *European conference on combinatorics, graph theory and applications 2021 (EuroComb 2021)*, 2021.

Rufus Isaacs. Infinite families of nontrivial trivalent graphs which are not tait colorable. *The American Mathematical Monthly*, 82(3):221–239, 1975. doi: 10.1080/00029890.1975.11993805.

Alon Itai and Michael Rodeh. Covering a graph by circuits. In *Lecture Notes in Computer Science*, volume 62, pages 289–299, 07 1978. doi: 10.1007/3-540-08860-1_21.

François Jaeger. Flows and generalized coloring theorems in graphs. *Journal of Combinatorial Theory, Series B*, 26(2):205–216, 1979. doi: 10.1016/0095-8956(79)90057-1.

François Jaeger, Nathan Linial, Charles Payan, and Michael Tarsi. Group connectivity of graphs—a nonhomogeneous analogue of nowhere-zero flow properties. *Journal of Combinatorial Theory, Series B*, 56(2):165–182, 1992. doi: 10.1016/0095-8956(92)90016-Q.

Brian W. Kernighan and Dennis M. Ritchie. *The C programming language*. 1988.

Martin Kochol. A cyclically 6-edge-connected snark of order 118. *Discret. Math.*, 161:297–300, 1996a.

Martin Kochol. Snarks without small cycles. *J. Comb. Theory, Ser. B*, 67:34–47, 1996b.

Martin Kochol. Superposition and constructions of graphs without nowhere-zero k-flows. *European Journal of Combinatorics*, 23(3):281 – 306, 2002. doi: 10.1006/eujc.2001.0563.

Martin Kochol. Reduction of the 5-flow conjecture to cyclically 6-edge-connected snarks. *Journal of Combinatorial Theory, Series B*, 90(1):139–145, 2004. doi: 10.1016/S0095-8956(03)00080-7.

Martin Kochol. Restrictions on smallest counterexamples to the 5-flow conjecture. *Comb.*, 26(1):83–89, 2006. doi: 10.1007/s00493-006-0006-1.

Martin Kochol. Smallest counterexample to the 5-flow conjecture has girth at least eleven. *Journal of Combinatorial Theory, Series B*, 100(4):381–389, 2010. doi: 10.1016/j.jctb.2009.12.001.

Daniel Kráľ. Group coloring is Π_2^P-complete. *Theor. Comput. Sci.*, 349(1):99–111, December 2005. doi: 10.1016/j.tcs.2005.09.033.

Daniel Kráľ and Pavel Nejedlý. *Group Coloring and List Group Coloring Are Π_2^P-Complete*, pages 274–286. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. doi: 10.1007/978-3-540-28629-5_19.

Hong-Jian Lai, Xiangwen Li, Yehong Shao, and Mingquan Zhan. Group connectivity and group colorings of graphs - a survey. *Acta Mathematica Sinica, English Series*, 27:405–434, 03 2011. doi: 10.1007/s10114-010-9746-3.

Jiaao Li, 2018. Personal communication.

C. H. C. Little and R. D. Ringeisen. On the strong graph embedding conjecture. In *Proceeding of the 9th Southeastern Conferenceo Combinatorics, Graph Theory arid Computing*, pages 479–487, Winnipeg, 1978.

László M. Lovász, Carsten Thomassen, Yezhou Wu, and Cun-Quan Zhang. Nowhere-zero 3-flows and modulo k-orientations. *J. Combin. Theory Ser. B*, 103(5):587–598, 2013. doi: 10.1016/j.jctb.2013.06.003.

Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60:94–112, 2014. doi: 10.1016/j.jsc.2013.09.003.

Bojan Mohar and Carsten Thomassen. *Graphs on Surfaces*. Johns Hopkins series in the mathematical sciences. Johns Hopkins University Press, 2001.

Bojan Mohar and Andrej Vodopivec. On polyhedral embeddings of cubic graphs. *Combinatorics, Probability and Computing*, 15(6):877–893, 2006.

Lucie Mohelníková. Group connectivity of graphs. Master's thesis, Charles University in Prague, Faculty of Mathematics and Physics, 2014. URL https://is.cuni.cz/webapps/zzp/detail/148945/?lang=en. [in Czech].

Michael Molloy and Mohammad R. Salavatipour. A bound on the chromatic number of the square of a planar graph. *Journal of Combinatorial Theory, Series B*, 94(2):189–213, 2005. doi: 10.1016/j.jctb.2004.12.005.

Crispin A. Nash-Williams. Edge-disjoint spanning trees of finite graphs. *Journal of the London Mathematical Society*, s1-36(1):445–450, 1961. doi: 10.1112/jlms/s1-36.1.445.

James G. Oxley. *Matroid Theory (Oxford Graduate Texts in Mathematics)*. Oxford University Press, Inc., USA, 2006. doi: 10.5555/1197093.

Julius Petersen. Sur le théorème de tait. *L'Intermédiaire des Mathématiciens*, 1898.

Robert W. Robinson and Nicholas C. Wormald. Almost all cubic graphs are hamiltonian. *Random Struct. Algorithms*, 3(2):117–125, March 1992. doi: 10.1002/rsa.3240030202.

Alexander Schrijver. Characterizing partition functions of the edge-coloring model by rank growth. *Journal of Combinatorial Theory, Series A*, 136:164–173, 2015. doi: 10.1016/j.jcta.2015.06.007.

Paul D. Seymour. Sums of circuits. In *Graph Theory and Related Topics*, page 342–355, 1979.

Paul D. Seymour. Nowhere-zero 6-flows. *Journal of Combinatorial Theory, Series B*, 30(2):130 – 135, 1981. doi: 10.1016/0095-8956(81)90058-7.

Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley Professional, 4th edition, 2013.

Balázs Szegedy. Edge coloring models and reflection positivity. *Journal of the American mathematical Society*, 20(4):969–988, 2007. doi: 10.1090/S0894-0347-07-00568-1.

George Szekeres. Polyhedral decompositions of cubic graphs. *Bulletin of the Australian Mathematical Society*, 8(3):367–387, 1973. doi: 10.1017/S0004972700042660.

Peter G. Tait. *Note on a Theorem in Geometry of Position*. Royal Society of Edinburgh, 1880.

The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 9.2)*, 2021. `https://www.sagemath.org`.

Robin Thomas. Recent excluded minor theorems for graphs. In *Surveys in Combinatorics*, pages 201–222. Univ. Press, 1999.

Carsten Thomassen. On the number of hamiltonian cycles in bipartite graphs. *Combinatorics, Probability and Computing*, 5(4):437–442, 1996. doi: 10.1017/S0963548300002182.

Carsten Thomassen. Many 3-colorings of triangle-free planar graphs. *J. Combin. Theory Ser. B*, 97(3):334–349, 2007a. doi: 10.1016/j.jctb.2006.06.005.

Carsten Thomassen. Exponentially many 5-list-colorings of planar graphs. *Journal of Combinatorial Theory, Series B*, 97(4):571–583, 2007b. doi: 10.1016/j.jctb.2006.09.002.

Carsten Thomassen. The weak 3-flow conjecture and the weak circular flow conjecture. *Journal of Combinatorial Theory, Series B*, 102(2):521–529, 2012. doi: 10.1016/j.jctb.2011.09.003.

Heinrich Tietze. Einige bemerkungen zum problem des kartenfärbens auf einseitigen flächen. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 1910.

William T. Tutte. On the imbedding of linear graphs in surfaces. *Proceedings of the London Mathematical Society*, s2-51(1):474–483, 1949. doi: 10.1112/plms/s2-51.6.474.

William T. Tutte. A contribution to the theory of chromatic polynomials. *Canadian J. Math.*, 6:80–91, 1954.

William T. Tutte. On the problem of decomposing a graph into n connected factors. *Journal of the London Mathematical Society*, s1-36(1):221–230, 1961. doi: 10.1112/jlms/s1-36.1.221.

William T. Tutte. A geometrical version of the four color problem. *Combinatorial Mathematics and its Applications*, 1967.

Guido van Rossum and Fred L. Drake. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

Guido van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

Hassler Whitney. Congruent graphs and the connectivity of graphs. *American Journal of Mathematics*, 54(1):150–168, 1932. URL http://www.jstor.org/stable/2371086.

Helmut Wielandt. Das iterationsverfahren bei nicht selbstadjungierten linearen eigenwertaufgaben. *Mathematische Zeitschrift*, 50(1):93–143, Dec 1944. doi: 10.1007/BF01312438.

Cun-Quan Zhang. *Integer flows and cycle covers of graphs*, volume 205 of *Monographs and Textbooks in Pure and Applied Mathematics*. Marcel Dekker, Inc., New York, 1997.

Cun-Quan Zhang. *Circuit Double Cover of Graphs*. London Mathematical Society Lecture Note Series. Cambridge University Press, 2012. URL https://books.google.cz/books?id=0cM0AAAAQBAJ.

Xuding Zhu. Circular chromatic number: a survey. *Discrete Mathematics*, 229 (1-3):371–410, 2001. doi: 10.1016/S0012-365X(00)00217-X.

List of Figures and Other Floats

List of Publications

1. Ondřej Čepek and Radek Hušek. Recognition of tractable DNFs representable by a constant number of intervals. *Discrete Optimization*, 2016. doi: 10.1016/j.disopt.2016.11.002.

2. Radek Hušek, Peter Korcsok and Robert Šámal. On girth of minimal counterexample to 5-flow conjecture. In *Bordeaux Graph Workshop*, 2016.

3. Radek Hušek, Lucie Mohelníková, and Robert Šámal. Group connectivity: $\mathbb{Z}_4$ vs $\mathbb{Z}_2^2$. *Journal of Graph Theory*, 2019. doi: 10.1002/jgt.22488.

   Also in *Bordeaux Graph Workshop*, 2016.

4. Radek Hušek, Dušan Knop, and Tomáš Masařík. Approximation algorithms for Steiner tree based on star contractions: A unified view. In *15th International Symposium on Parameterized and Exact Computation (IPEC 2020)*, 2020. ISBN 978-3-95977-172-6. doi: 10.4230/LIPIcs.IPEC.2020.16.

5. Radek Hušek and Robert Šámal. Homomorphisms of Cayley graphs and cycle double covers. *The Electronic Journal of Combinatorics*, 2020. doi: 10.37236/8456.

   A partial version also in *European conference on combinatorics, graph theory and applications 2017 (EuroComb 2017)*.

6. Radek Hušek and Robert Šámal. Counting Circuit Double Covers. In *European conference on combinatorics, graph theory and applications 2021 (EuroComb 2021)*, 2021.