Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Algorithm to determine the circular chromatic index of a cubic graph

Bachelor Thesis

2024
Radoslav Petráni

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Algorithm to determine the circular chromatic index of a cubic graph

Bachelor Thesis

Study Programme:	Computer Science
Field of Study:	Computer Science
Department:	Department of Computer Science
Supervisor:	doc. RNDr. Robert Lukoťka, PhD.

Bratislava, 2024
Radoslav Petráni

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Radoslav Petráni
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Algorithm to determine the circular chromatic index of a cubic graph
Algoritmus na určenie cirkulárneho chromatického indexu kubického grafu.

Anotácia: Cirkulárne hranové r-farbenie grafu je priradenie farieb z intervalu [0, r) hranám graph tak, že rozdiel hodnôt na susedných hranách je z intervalu [1, r-1].
Cirkulárny chromatický index grafu je infímum zo všetkých hodnôt r, pre ktoré má graf cirkulárne hranové r-farbenie.
Cieľom práce je navrhnúť algoritmus na výpočet tohoto invariantu.
Pojem toku balancovaného ohodnotenia [Samuel Vavrek, Algoritmus na určenie cirkulárneho tokového čísla, diploma thesis, FMFI UK, 2023] ukazuje, ako efektívne zakódovať duálny koncept cirkulárneho toku ak zmiešaný lineárny program. Tento prístup možno čiastočne zovšeobecniť na problém existencie r-napätia (r-tension), ktorý je duálny k cirkulárnym r-farbeniam (vrcholovým), aj na plochách iných ako rovina.

Vedúci: doc. RNDr. Robert Lukoťka, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: prof. RNDr. Martin Škoviera, PhD.

Spôsob sprístupnenia elektronickej verzie práce:
bez obmedzenia

Dátum zadania: 16.10.2023

Dátum schválenia: 16.10.2023 doc. RNDr. Dana Pardubská, CSc.
garant študijného programu

.....
študent

.....
vedúci práce

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Radoslav Petráni
Study programme:	Computer Science (Single degree study, bachelor I. deg., full time form)
Field of Study:	Computer Science
Type of Thesis:	Bachelor´s thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Algorithm to determine the circular chromatic index of a cubic graph

Annotation: Circular r-edge-colouring of a graph is an assignment of of colours from interval [0, r) to the edges of the graph in such a way, that the difference of colours of neighbouring edges is in [1, r-1].
Circular chromatic index of a graph is the infimum over all r's such that the graph has a circular r-edge-colouring.
The goal of the thesis is to propose and implement an algorithm to compute this invariant on cubic graph.
The concept of balanced valuation flows [Samuel Vavrek, Algoritmus na určenie cirkulárneho tokového čísla, diploma thesis, FMFI UK, 2023] shows how to encode the dual concept of circular flow efficiently as an mixed integer linear programming problem.
This approach can be, to a degree, generalised to r-tensions, which are dual to circular r-colourings (vertex) on surfaces other than the plane.

Supervisor:	doc. RNDr. Robert Lukoťka, PhD.
Department:	FMFI.KI - Department of Computer Science
Head of department:	prof. RNDr. Martin Škoviera, PhD.
Assigned:	16.10.2023
Approved:	16.10.2023

doc. RNDr. Dana Pardubská, CSc.
Guarantor of Study Programme

.....
Student

.....
Supervisor

Acknowledgments: I would like to thank my supervisor doc. RNDr. Robert Lukoťka, PhD. for his help and guidance with this thesis.

Abstrakt

Cirkulárne r-hranové farbenie grafu $G = (V, E)$ je zobrazenie $c : E \to [0, k)$, kde pre každe dve susedné hrany e_1 a e_2, $1 \leq |c(e_1) - c(e_2)| \leq k - 1$ a cirkulárny chromatický index grafu je infímum z r, takých, že graf je cirkulárne r-hranovo zafarbiteľný. Cirkulárny r-tok je priradenie orientácie a tokovej funkcie: $\phi : E \to [1, r - 1]$ grafu tak, že súčet tokových hodnôt hrán vchádzajúcich a vychádzajúcich sa musí rovnať pre každý vrchol grafu.

Existuje dualita medzi farbeniami a nikde-nulovými tokmi. Podobná dualita existuje medzi cirkulárnymi farbeniami a r-napätiami, ktoré definoval DeVos.

Na výpočet možno použiť koncept balancovaného ohodnotenia r-tokov na zakódovanie r-napätia ako zmiešaný lineárny program.

Túto dualitu sme využili na vytvorenie algoritmu, ktorý počíta cirkulárny chromatický index grafu. Potom porovnáme dobu behu tohto algoritmu a triviálneho algoritmu na výpočet cirkulárneho chromatického indexu, ktorý používa zmiešané lineárne programovanie.

Kľúčové slová: cirkulárne chromatické farbenie, balancované ohodnotenie r-tok, zmiešané lineárne programovanie

Abstract

A r-circular edge coloring of a graph $G = (V, E)$ is a mapping $c : E \to [0, k)$, where for any two adjacent vertices e_1 and e_2, $1 \leq |c(e_1) - c(e_2)| \leq k - 1$ and the circular chromatic index is the infimum over all r, such that G has a r-circular edge coloring. A circular r-flow is an asigning of an orientation and a flow function $\phi : E \to [1, r - 1]$ to the graph, where the flow value of outgoing edges must be equal to the flow value of incoming edges for every vertex of the graph.

There exists a duality between colorings and nowhere zero flows. Similar duality exists between circular colorings and r-tensions, that were defined by DeVos.

The concept of balanced valuation r-flows can be used to compute r-tensions using mixed linear programming solvers.

We make use of this duality to construct an algorithm that computes the circular chromatic index of a graph. We will then compare the runtime of this algorithm to a trivial algorithm for computing circular chromatic index that uses mixed linear programming.

Keywords : circular chromatic index, balanced valuation flows, mixed integer linear programming

Contents

List of Tables

Introduction

This thesis is about finding an efficient method of finding circular chromatic indexes of graphs, mainly cubic graphs as they are the focus of our research group and many atributes of cubic graphs can be broadenend to graphs as a whole.

This work will be centered around circular edge coloring, we'll however first define circular colorings.

Circular coloring is a modification on the usual graph coloring, where instead of integers we map vertices to real numbers instead of integers.

Definition 1. *A k-circular coloring of a graph $G = (V, E)$ is a mapping $c : V \to [0, k)$, where for any two adjacent vertices u and v*

$$1 \leq |c(u) - c(v)| \leq k - 1$$

The circular chromatic number of a graph G, $\chi_c(G)$, is the infimum of all k, where G has a k-circular coloring. We say that a graph G is circularly k-colorable if there exists a circular k-coloring of G.

The circular chromatic number is well-defined, if a graph G has a circular k-coloring, then it has circular l-coloring for any $l > k$.

It was originaly defined by Vince in 1988 who called it star coloring [19].

Theorem 2. *[19] For any graph G,*

$$\chi(G) - 1 < \chi_c(G) \leq \chi(G)$$

.

Circular coloring is an often researched field of Graph Theory, here are some interesting articles conserning it. [23, 21, 22, 11, 4]

We'll be mostly focusing on edge circular colorings:

Definition 3. *A k-circular edge coloring of a graph $G = (V, E)$ is a mapping $c : E \to [0, k)$, where for any two adjacent vertices e_1 and e_2*

$$1 \leq |c(e_1) - c(e_2)| \leq k - 1$$

Definition 4. *The circular chromatic index of a graph G, $\chi'_c(G)$, is the infimum of all k, where G has a k-circular edge coloring.*

As deciding whether an integer is a graphs chromatic number (or an index) is an NP-complete problem [10], so is the problem of circular colorings. Thus finding an effective algorithm, could vastly improve the speed of algorithmic computation of the problem. Currently:

Kunertová [13] designed multiple methods to determine circular edge colorability and compared their time complexity and running time. Then she used these algorithms to compute circular indexes for small cubic snarks.

For snarks of orders higher then 20, dependant on the machine, this can take even a couple of minutes, we would like something better, that could make computing the indexes even of bigger graphs possible in an acceptable time.

We want to try one specific approach, where we will be transforming the graph into another, making use of the flow-coloring duality, such that the graph's index will be equivalent to the flow of the new graph.

A circular k-flow is an asigning an orientation, a flow function $\phi : E \to [1, k-1]$ to the graph, where the flow value of outgoing edges must be equal to the flow value of incoming edges for every vertex of the graph.

When we look at planar graphs, there does exist a duality between flows and colorings.

The duality first proven by Tutte(1954):

Theorem 5. *[3] For every dual planar multigraphs G and G'*

$$\chi(G) = \phi(G')$$

Similar duality exists between Circular coloring and circular flows. The proof of Tutte's duality Theorem can also be used as the proof of duality between circular Colorings and circular flows.

For non-planar orientable surfaces a similar relationship can be established. Circular coloring of a graph embedded in an orientable surface corresponds to a circural flow in a dual graph, as long as we ensure that the flow value on the edges dual to the irreducible cycles of the original graph, adds up to zero as well as all the conditions of the original duality are satisfied.

Balanced valuations r-flows defined by Lukoťka[15], based on the concept of balanced valuations by Jaeger [9], is the assignment of orientation to the graph and values from the interval $[0, \frac{r-2}{r}]$ to the edges of the graph so that the sum of outgoing edges minus the sum of incoming edges for each vertex is equal to the degree of the vertex modulo 2. This concept corresponds to circular r-flows and we will be using it to improve the calculation of the circular flow number.

Chapter 1

Circular coloring and its duality

1.1 Graphs with known chromatic indexes

Theorem 6 (Vizing [3]). *For every simple graph G:*

$$\Delta(G) \leq \chi'(G) \leq \Delta(G) + 1$$

Based on Vizings theorem we separate graphs into two classes, class one graphs that can be colored using $\Delta(G)$ colors and class two graphs that can be colored using $\Delta(G) + 1$ colors. In this thesis we are mainly interested in class two graphs.

First we'll look at some graphs with known circular chromatic indexes. We will be mainly looking at *snarks, cubic graphs, that are non-trivialy not 3-edge-colorable.* That is to say, they are class two graphs and are connected and bridgeless as graphs with bridges are always not 3-colorable. We also demand that the graph contains no triangles, as the triangle can be simplified while retaining its edge colorability by taking the vertices of the triangle and contracting them into a single vertex. Graphs containing a 4-cycle are also excluded as we can replace the cycle with two paralel edges. We therefore consider a graph snark if it is bridgeless and of girth at least 5.

In this work we will be using some of these snarks for the purpose of testing the correctness of our algorithm.

Mazák [17] talked about circular chromatic index of snarks by non-trivialy we exclude cubic graphs with bridges and graphs of low girth. He establishes the circular chromatic index of Petersen graph, which is $\frac{11}{3}$.

The main aim of Mazák's paper was to determine the chromatic circular index of Blanuša snarks, discovered by Danilo Blanuša in 1946, wich Petersen graph is a part of. He proves that circular chromatic index of B_m^1, which are Blanuša snarks created from m A-pieces(the basic piece of a graph used in the construction of Blanuša snarks) is:

$$\chi_c'(B_m^1) = 3 + \frac{2}{3m}$$

Another group of snarks with known circular chromatic indexes are the Flower snarks, first constructed by Isaacs, and Goldberg snarks discovered by Goldberg. The circular chromatic idnex of Flower snarks was defined by Ghebleh et. al [5] and Goldberg snarks by Ghebleh [6].

Lukoťka and Mazák [16] prove that for every r, $3 < r < 3 + \frac{1}{3}$ there exists a family of cubic graphs with circular chromatic index r.

1.2 Circular flows

Next we will be taking a look at flows and the duality that exists between them and colorings(and also circular flows and colorings).

Definition 7. *For directed multigraph $G = (V, E)$ and Abelian group A, $\phi : E \to A$ is a nowhere zero flow if for every $v \in V$:*

$$\sum_{e \in E^+(v)} \phi(e) = \sum_{e \in E^-(v)} \phi(e)$$

Where $\phi(e)$ is the flow function, $E^+(v)$ represents the set of edges incoming to v and $E^-(v)$ denotes outgoing edges of v. This condition is reffered to as Kirchoff's law,
And $\phi(e) \neq 0$ for every $e \in E$ (therefore nowhere zero).
If k is an integer and $0 < |\phi(e)| < k$ then ϕ is refered to as a nowhere zero k-flow. [3]
The flow number of graph G, $\phi(G)$, is the infimum of all integers r, such that G has a circular nowhere zero r-flow.

As we are working with real numbers, we will be using circular flows, a modification on nowhere zero flows, where we asign real values from $[1, r-1]$ to edges, such that Kirchoff's law applies. Circular flow number of graph G, $\phi_c(G)$, is the infimum of all $r \in \mathbb{R}$, such that G has a circular nowhere zero r-flow.

According to Tutte's 5-flow conjecture[3], every bridgeless multigraph has a 5-flow, therefore it also has a 5-circular flow. P.D.Seymore has proven in 1981 [18], that every bridgeless multigraph has a 6-flow.

1.3 Balanced valuations

Definition 8. *A balanced valuation r-flow on graph G is (O, ϕ, b), where O is an orientation, $b : V(G) \to \mathbb{Z}$, $\phi : E(G) \to \mathbb{R}$, such that:*

1. *$b(v) \equiv deg(v) (mod\ 2)$ for every vertex $v \in V(G)$,*
2. *$-\frac{r-2}{r} \leq \phi(e) \leq \frac{r-2}{r}$ for every edge $e \in E(G)$,*

3. $\sum_{e\in E^+(v)} \phi(e) - \sum_{e\in E^-(v)} \phi(e) = b(v)$ *for every vertex* $v \in V(G)$.

we can represent the circular r-flow more broadly as:

$$a_{11}x_1 + a_{12}x_2 \cdots + a_{1n}x_n = k_1 r$$
$$a_{21}x_1 + a_{22}x_2 \cdots + a_{2n}x_n = k_2 r$$
$$\vdots$$
$$a_{m1}x_1 + a_{m2}x_2 \cdots + a_{mn}x_n = k_m r$$

where every $a_{ij} \in \{-1, +1, 0\}$ represents the orientation of j-th edge, in relation to vertex i, $+1$ for incoming edges, -1 for outgoing ones and 0 if the edge is not incident to i. Also for every i, the following applies, $1 \leq x_i \leq r-1$. This way we can later, instead of vertices, also include the irreducible cycles of the graph into this definition.

As for balanced valuation r-flows we can similarly define them as:

$$a_{11}y_1 + a_{12}y_2 \cdots + a_{1n}y_n = b_1$$
$$\vdots$$
$$a_{m1}y_1 + a_{m2}y_2 \cdots + a_{mn}y_n = b_m$$

where every $b_i \equiv \sum_j a_{ij} (mod\ 2)$ and for every i applies $-\frac{r-2}{2} \leq y_i \leq \frac{r-2}{2}$.

Theorem 9. *A bridgeless graph has a circular nowhere zero r-flow, if an only if it has a balanced valuation r-flow.*

proof:

"⇒" From formulas $-\frac{r-2}{2} \leq y_i \leq \frac{r-2}{2}$ and $1 \leq x_i \leq r-1$, we get:

$$1.\ y_i = (x_i \frac{2}{r} - 1)$$

and

$$2.\ x_i = (y+1)\frac{r}{2}$$

we then subtitute 2. into the first set of equations, the ones for circular flows:

$$a_{i1}x_1 + a_{i2}x_2 \cdots + a_{in}x_n = k_i r$$
$$a_{i1}(y_1+1)\frac{r}{2} + a_{i2}(y_2+1)\frac{r}{2} \cdots + a_{in}(y_n+1)\frac{r}{2} = k_i r$$
$$\frac{r}{2}(a_{i1}(y_1+1) + a_{i2}(y_2+1) \cdots + a_{in}(y_3+1)) = k_i r$$
$$a_{i1}(y_1+1) + a_{i2}(y_2+1) \cdots + a_{in}(y_n+1) = 2k_i$$

$$a_{i1}y_1 + a_{i2}y_2 \cdots + a_{in}y_n + \sum_{j\in(1,n)} a_{ij} = 2k_i$$

$$a_{i1}y_1 + a_{i2}y_2 \cdots + a_{in}y_n = 2k_i - \sum_{j\in(1,n)} a_{ij}$$

parity of $2k_i - \sum_{j\in(1,n)} a_{ij}$, is the same as the parity of just the sum and both $2k_i$ and the sum are integers. Therefore $2k_i - \sum_{j\in(1,n)} a_{ij} = b_i$.

"$\Leftarrow$" we substitute 1. into the second set of equations:

$$a_{i1}y_1 + a_{i2}y_2 \cdots + a_{in}y_n = b_i$$

$$a_{i1}(x_1\frac{2}{r} - 1) + a_{i2}(x_2\frac{2}{r} - 1) \cdots + a_{in}(x_3\frac{2}{r} - 1) = b_i$$

$$a_{i1}x_1\frac{2}{r} + a_{i2}x_2\frac{2}{r} \cdots + a_{in}x_3\frac{2}{r} - \sum_{j\in(1,n)} a_{ij} = b_i$$

$$\frac{2}{r}(a_{i1}x_1 + a_{i2}x_2 \cdots + a_{in}x_3) = b_i + \sum_{j\in(1,n)} a_{ij}$$

$$a_{i1}x_1 + a_{i2}x_2 \cdots + a_{in}x_3 = (b_i + \sum_{j\in(1,n)} a_{ij})\frac{r}{2}$$

As b_i and $\sum_{j\in(1,n)} a_{ij}$ have the same parity, $(b_i + \sum_{j\in(1,n)} a_{ij})/2$ is always an integer, we'll call it k_i, so:

$$a_{i1}x_1 + a_{i2}x_2 \cdots + a_{in}x_3 = k_i r$$

□

Existing algorithms for circular flow numbers

Goedgeburg et al. [7] came up with an algorithm for for the computation of the circular flow numbers of bridgeless cubic graphs. They then, using this algorithm, determined the flow numbers for all snarks up to 36 vertices and that of various famous snarks.

Lukoťka describes [14] efficient polynomial algorithms to turn balanced valuations and orientations into circular nowhere zero r-flows.

Lukoťka [15] also computed circular nowhere-zero flows, using balanced valuation r-flows and mixed linear programming solvers. Using this method it was possible to compute flow-numbers of cubic graphs of order up to 150, in a reasonable time of running the algorithm of less then 15 hours.

1.4 Duality between flows and colorings

Our interest in flows lies in the existence of a duality between nowhere zero flows and colorings, which we will be using in our algorithm.

Definition 10 ([20])**.** *An embedding of a graph G on a surface Σ is a representation of G on Σ in which points of Σ are associated with vertices and simple arcs are associated with edges, such that:*

- *the endpoints of the arc associated with an edge e are the points associated with the endvertices of e*
- *no arcs include points associated with other vertices*
- *two arcs never intersect at a point which is interior to either of the arcs*

Definition 11. *Let G be an embedded graph, we define G', a graph in the same surface, as its dual graph. The vertices of G' are the faces of G. For every edge $e \in E(G)$ there exists an edge $e' \in E(G')$ that connects the two faces, that were adjacent to e.*

As the flow-coloring duality applies to vertex colorings, we will be using the line graph of the original graph instead,

Theorem 12. *Line graph L(G) of G, is a such a graph that:*

- *each vertex of L(G) represents an edge in G*
- *two vertices of L(G) are adjacent, if their edges in G were incident.*

In this way we represent the edges of the original graph as vertices in the line graph and we can continue using circular coloring instead of edge coloring on the original graph.

Definition 13. *circular chromatic index of a graph G is the circular chromatic number of its line graph $L(G)$:*

$$\chi'(G) = \chi(L(G))$$

We may then continue with circular vertex coloring using the line graph instead and make use of its properties.

Theorem 14 (Euler formula,[3])**.** *for every connected planar graph G:*

$$|V(G)| - |E(G)| + |F(G)| = 2$$

For graphs that can be embedded in surfaces, the *Euler characteristic* χ is defined:

$$\chi = |V(G)| - |E(G)| + |F(G)|$$

thus, every connected planar graph has the Euler characteristic of 2. A Genus, g of a orientable surface can be defined using the Euler characteristic, $\chi = 2 - 2g$. A genus 0 orientable surface is a sphere, genus 1 is a torus, genus 2 a double torus etc.

Definition 15. *A cycle is a connected graph, where the degree of every vertex is even. A circuit is a connected graph, where the degree of every vertex is 2.*

Definition 16. *For a given graph G, edge space of G is the set of all subsets of $E(G)$. This set forms a vector space over the $\mathbb{Z}_2$ field, where for every edge of $E(G)$, 0 represent the absence of the edge in the subset and 1 its presence.*

Definition 17 ([8])**.** *The cycle space of a Graph G is the subset of the Edge space consisting of graph $\emptyset$, all the cycles in G, and all unions of edge-disjoint cycles of G*

The flow-coloring duality theorem, however works under the condition that the graph, and such also its dual, are planar. What about graphs that are not planar? Can we define some sort of duality between them and nowhere zero flows? We will be taking a look at graphs embedded in different surfaces than planes.

DeVos et al. define the concept of tension and local tension as well as i-chains, boundries and coboundries.

Definition 18 ([2])**.** *For directed graph G embedded in a surface. Map Φ: $E(G) \to \mathbb{R}$ is a tension if for every $C \subset G$, if the sum of the forward edges of C is equal to the sum of backward edges of C. If this is satisfied for every contractible curve in the surface, Φ is a local tension.*

This concept of tensions corresponds to our concept of circular chromatic colorings.

Theorem 19 ([2])**.** $\chi_c(G) = inf\{\alpha \in \mathbb{R} | G \text{ admits an } \alpha - tension\}$

Similarly we can define for local tensions:

Definition 20. $\chi_{loc}(G) = inf\{\alpha \in \mathbb{R} | G \text{ admits a local } \alpha - tension\}$
We call $\chi_{loc}(G)$ the local circular chromatic number of G.

Lets take an Abelian group A and an embedded graph G . We call *0-chain* a map from V(G) to A, we call *1-chain* a map from E(G) to A, and *2-chain* a map F(G) to A. Let $i = 0, 1, 2,$, then $i-$chains form a group under componentwise addition, we denote $C_i(G, A)$.

We define $< v, e >$ for vertex $v \in V(G)$ and edge $e \in E(G)$ as $< v, e >= 0$ if v and e are not incident, as $< v, e >= -1$ if v is the first vertex of e and as $< v, e >= 1$ if it is the second edge. For a face F and its bounding cycle $S = v_0, e_1, v_1, \ldots, e_k, v_k$ we define $< e, F >= \sum_{1 \le i \le k | e = e_i} < v_i, e_i >$.

Definition 21 ([2]). *If $c \in C_0(G, A)$, we define the coboundry of c as $\delta c \in C_1(G, A)$ where $\delta c(e) = \Sigma_{v \in V(G)} < v, e > c(v)$.*

If $c \in C_1(G, A)$, we define the coboundry of c as $\delta c \in C_2(G, A)$ where $\delta c(F) = \Sigma_{e \in E(G)} < e, F > c(e)$.

If $c \in C_1(G, A)$, we define the boundry of c as $\partial c \in C_0(G, A)$ where $\partial c(F) = \Sigma_{e \in E(G)} < v, e > c(e)$.

If $c \in C_2(G, A)$, we define the boundry of c as $\partial c \in C_1(G, A)$ where $\partial c(e) = \Sigma_{F \in F(G)} < e, F > c(F)$.

If c^1 is a 1-chain and $c^1 = \delta c^0$ for 0-chain c^0, we call c^1 a tension. We note the set of tensions as $T(G, \Gamma)$.

If $\partial c^1 = 0$, c^1 is a flow. We note the set of flows as $L(G, \Gamma)$.

If G is an embedded graph and $\delta c^1 = 0$, we call c^1 a local tension. We note the set of local tensions as $F(G, \Gamma)$.

If $c^1 = \partial c^2$ for $c^2 \in C_2(G, \Gamma)$, we call c^1 a facial flow. We note the set of facial flows as $K(G, \Gamma)$.

$C_1(G, \mathbb{Q})$ is a vector space. $F(G, \mathbb{Q})$ and $K(G, \mathbb{Q})$ are its subspaces. Also, $K(G, \mathbb{Q})$ is a subspace of $F(G, \mathbb{Q})$.

Lemma 22. *Let G be a graph embedded in a surface. Let $A_1, \dots A_k$ be an integer basis of vector space $K(G, \mathbb{Q})$. Let $A_1, \dots A_k, B_1, \dots B_m$ be an integer basis of vector space $F(G, \mathbb{Q})$, then $x \in T(G, \mathbb{R}) \Leftrightarrow x \in L(G, \mathbb{R})$ and $x.B_i = 0$ for all $i \in \{1, \dots m\}$.*

proof (R. Lukoťka, personal communication):
$\Rightarrow$: See [2, propossition 3.1]
$\Leftarrow$: Let x be a flow, $x \in L(G, \mathbb{R})$ and $x.B_i = 0$ for all $i \in \{1, \dots m\}$, as $x \in L(G, \mathbb{R})$, $x.A_i = 0$ for all $i \in \{1, \dots k\}$.
We want to prove, that $x \in T(G, \mathbb{R})$, based on propossition 3.1, we need to prove: $d.x = 0$ for every $d \in F(G, \mathbb{Z})$. Let us take any $d \in F(G, \mathbb{Z})$.
Every flow $d \in F(G, \mathbb{Z})$ is also a flow in $F(G, \mathbb{Q})$, so

$$d = y_1.A_1 + \dots y_k.A_k + z_1.B_1 + \dots + z_m.B_m$$

where y_i and z_j are from Q.
We multiply by lowest common denominator:

$$z.d = y'_1.A_1 + \dots y'_k.A_k + z'_1.B_1 + \dots + z'_m.B_m$$

where y'_i and z'_j are from Z.

$$z.(d.x) = (z.d).x = (y'_1.A_1 + \dots y'_k.A_k + z'_1.B_1 + \dots + z'_m.B_m).x =$$
$$= (y'_1.A_1).x + \dots (y'_k.A_k).x + (z'_1.B_1).x + \dots + (z'_m.B_m).x = 0$$

In $\mathbb{R}, z.(d.x) = 0$, means $d.x = 0$. □

Therefore if a graph could be represented as embedded in an orientable surface, we could, and it is our aim in this paper, transform it into its dual and calculate its circular chromatic index using the dual's flow number.

In [12] the authors looked at some interesting characteristics of high girth graphs, that is graph that don't have any short cycles. They prove that the circular chromatic index of every cubic graph with girth 6 or more is less or equal than $\frac{7}{2}$ and that the index of a subcubic graph with odd-girth at least 7 is at m ost $\frac{7}{2}$.

Chapter 2

Implementation

In this chapter, we will explain the process through which we will transform a graph to its dual, so that we may use the linear program for its flow number, and describe the basic program that calculates circular chromatic numbers.

2.1 Trivial circular coloring algorithm

First we will describe the aformentioned algorithm that uses mixed linear programming to calculate the circular chromatic index of a given cubic graph.

While this algorithm may not be the most efficient method of solving this problem, it will give us a rough estimate as to what orders of cubic graphs a simple algorithm can handle and we can use this estimate as comparison to our algorithm.

For each edge e of graph G we set a variable x_e that will correspond to the color of e, $x_e = c(e)$. The graph is represented as a two dimentional array of vertex adjecency, there will be 2 such variables for each edge of the graph. To encode the circular chromatic index of the graph, χ'_c, we add a variable t.

We set the upper bound for t as 4, as chromatic index of a cubic graph is less then 4, as according to Vizings theorem [3] $\Delta(G) \leq \chi'(G) \leq \Delta(G) + 1$, and the circular index is always smaller or equal than its chromatic index, $\chi'_c(G) \leq \chi_c(G)$.

We will then set constraints for each vertex v such that for any two edges, e and f that are incident to v :

$$1 \leq |x_e - x_f| \leq t - 1,$$

We will also need 3 binary variables for each vertex,b_{v1}, b_{v2}, b_{v3}, that one for each pair of incident edges, to represent the absolute value. We want to represent a constraint with absolute value in linear programming. We'll split this formula into two parts, $|x_e - x_f| \leq t - 1$ and $|x_e - x_f| \geq 1$.

To represent the first inequality is easy, we just add two constraint,

$$x_e - x_f \leq t - 1$$

$$-(x_e - x_f) \leq t - 1$$

as t-1 is always positive and either $x_e - x_f$ or $-(x_e - x_f)$ is always negative.

To represent $|x_e - x_f| \geq 1$ we will need to make use of the binary variables. We'll add two constraints:

$$(x_e - x_f) + M.b_{e,f} \geq 1$$

$$-(x_e - x_f) + M.(1 - b_{e,f}) \geq 1,$$

where $b_{e,f}$ is the binary variable for e and f and M is a constant large enough, that if $b = 1$, that inequality will be true independently of $x_e - x_f$, for cubic graphs we'll choose 4, as $x_e - x_f < 4$ based on Vizings theorem. Beacause of this the constraints will be, depending on $b_{e,f}$, either:

$$(x_e - x_f) \leq 1$$

$$-(x_e - x_f) + 4 \leq 1$$

where the second inequality is always true, or:

$$(x_e - x_f) + 4 \leq 1$$

$$-(x_e - x_f) \leq 1$$

where the first inequality is always true.

Lastly, for every edge e and its variable x_e we also need to set the bound:

$$x_e \in [0, t)$$

We will be running the program using the lp_solve5.5 mixed linear programming solver [1] . Using this algorithm, for cubic graphs of smaller orders, graphs with up to twenty vertices we are able to get results in a relativly short time. These results are of course dependent on hardware and other factors.

2.2 Duality algorithm

As we have said, our algorithm will be transforming a graph into its dual, while keeping the right orientation. We also need to asure, that the flow coloring duality of the graph is retained. Altough we start with a cubic graph, its line graph, so we can think of our aim as finding a vertex coloring for the cubic graph. The line graph won't be a cubic graph. It will be a 4-regular graph, as each edge in a cubic graph has 2 other edges incident to each of the vertices of this edge.

We won't analyze the time complexity of this algorithm, as this algorithm will be polynomial. The time complexity of the linear program is non-polynomial, so the

runtime of the whole program will be determined based on the effectivity of the linear algorithm.

The following algorithm will be written in c++ code, and this code will be then used when gathering results in the 3. chapter of this paper. We will however try to describe the algorithm independent of c++ syntax and the same principles should apply regardless of the choice of programming language, the effectivity of the algorithm could be affected and the times achieved by our algorithm in comparison to the trivial program may also be different.

Faces of the graph

First, for the construction of the dual graph and finding the basis of the cycle space of the graph, we will need to find the faces of the graph. Knowing the number of faces of the given graph and using the Euler formula ,Theorem 14, will allow us to find what surface can the given graph be embedded in.

To find the faces of a graph we use a simple algorithm. We start by finding an edge not yet used in any face. We then go to the second vertex of the edge, and find the next edge incident to this vertex in some given orientation of the surface. We then go to the second vertex of this edge and repeat the same. We do this until we return to the edge we started with. This sequence of edges is one of the faces. We then find another edge that wasn't yet used twice, as each edge is present twice in faces of the graph, it could even be twice in a single face. We repeat the same approach as the first time and continue this until all the edges of the graph has been accounted for.

We will be representing these faces as edge matrices. For this we need some ordering of edges. We'll pick any trivial ordering, we will be using this order for all edge matrices of this graph and later for the graphs dual, with the appropriate edges.

The edge matrix will belong to $\mathbb{Z}_2$. Value of 1 if the edge belongs to the face, or any other subgraph we will be using edge matrices for, and 0 otherwise. Next, the algorithm will transform the face matrix into a triangular matrix. The pseudocode for this algorithm is shown in algorithm 1.

basis of cycle space

If the graph is planar, the set of face cycles will form a basis of cycle space of the graph. In non-planar graphs, we will need to find the basis of cycle space and by finding which cycles are lineary independent to the face matrix, we will find those cycles that are irreducible in the non-planar embedding.

For the purpose of finding thebasis of cycle space of the graph, we'll next need to find a spanning tree of the graph. We will be using a trivial algorithm for finding a spanning tree of the graph. First we take an ordering of all the edges in the graph. We

Algorithm 1 Algorithm to transform face cycles into edge matrix

```
edges ← array of all edges of the graph
for face in faces do
    row ← new Array
    for edge i in edges do
        found ← 0
        for edge j in face do
            if i == j then
                found ← 1
            end if
        end for
        row.push ← found
    end for
    result.push ← row
end for
```

start with a spanning tree graph with only the first edge. We add the second edge in the ordering to our spanning tree. We then go through the spanning tree using breadth first search to see if any cycles exist in it. If a cycle exists, we remove the edge last added. If a cycle does not exist, we keep the edge in the spanning tree. We then add the next edge and repeat the same process. We do this until we have tried every edge or until the number of edges in the spanning tree is equal to the number of vertices minus one, as that means all vertices are included in the spanning tree. Note that this method will give us an arbitrary spanning tree, and for the purposes of effectivity of the mixed linear program it may not be the best approach.

A basis of cycle space can be made from a spanning tree, containing cycles formed by an edge e outside the spanning tree and the edges from the spanning tree that create the cycle from one vertex of e to the other. To find these cycles, we'll go through all the edges in the graph. If this edge is not present in the spanning tree, we'll run a function, that will, using breadth first search, find a path in the graph from one vertex of the edge to the other. This path together with the edge will be the cycle we were after. We'll repeat this approach with all the edges not in the spanning tree and we'll get a set of cycle that form the basis of the cycle space of the graph.

Irreducible cycles

Then, as by lemma 22, we have the basis of the facial flow space $A_1, \ldots, A_k$, that we can take from the faces of the graph and the basis of vector space $A_1, \ldots, A_k, B_1, \ldots, B_m$, that is the cycle basis from the spanning tree algorithm. We then need to find which

cycles are the irreducible cycles, that is cycles $B_1, \ldots, B_m$. We'll do this by transforming the cycle basis, tranforming it into an edge matrix, using the same algorithm as with the facial matrix.

This matrix, denoted as cycleMatrix, is a two dimentional $n * m$ array of elements from $\mathbb{Z}_2$, where each row represents one of the m cycles from the spanning tree algorithm and for each edge of the graph e_i, $i \in [0, n)$, $cycleMatrix[k][i] = 1$ if the k-th cycle contains the i-th edge and 0 otherwise.

We also have the facial Matrix, denoted as faceMatrix, a two dimentional $n * f$ array of elements from $\mathbb{Z}_2$, constructed the same way as the cycleMatrix. We have however already triangulated this matrix.

Next we'll take the facial matrix and one by one add the cycles, in the cycle basis, then after each we'll once again triangulate as seen in algorithm 2 and 3.

Algorithm 2 Algorithm for finding the irreducible cycles in a graph

```
for cycle c in cycleMatrix do
    originalSize ← faceMatrix.length
    faceMatrix.push ← c
    triangulate(faceMatrix)                          ▷ algorithm 3
    if faceMatrix.size > originalSize then
        result.push ← c
    end if
end for
```

The triagulation of a matrix as defined in algorithm 3 works followingly. We have an array, leadingOnes, where we keep track of which rows leading ones have already been determined. We then go column by column until we find a row that starts with one and its leading one has not been determined. This row will have the leading one in this column. We the add this row to all other rows that have a one in this column. After this operations the first column will have only one leading one and all other zeros. We continue this until all rows have asigned leading ones, then the matrix is in triangular form.

If the row added, that is the cycle, is lineary dependant, that would mean it was already part of the original facial basis and as such the cycle belongs to $A_1, \ldots, A_k$, not $B_1, \ldots, B_m$. After repeating this with all the rows of the matrix we get the set of edges not in the original basis, $B_1, \ldots, B_m$.

Orientation

For the purposes of computing the flow of the graph we will need to find the orientation of the graph. We'll pick the orientation for the edges of the graph even before the

Algorithm 3 Funcions that gets a matrix over $\mathbb{Z}_2$ and makes it a triangular matrix using row operations

```
procedure TRIANGULATE(matrix)
    array<bool> leadingOnes ← (matrix.size, false)
    rowsCompleted ← 0
    for i=0, i<matrix.columnNumber, i++ do
        j ← 0
        while j<matrix.size do
            if matrix[j][i]=1 and !leadingOnes[j] then
                leadingOnes[j] ← true;
                rowsCompleted++
                for k = 0, k < matrix.size, k++ do
                    if matrix[k][i]=1 and j!=k then
                        addRows(matrix[j], matrix[k])          ▷ addition (mod2)
                    end if
                end for
                break
            end if
            j++
        end while
        if rowsCompleted=matrix.size then
            break
        end if
    end for
    removeZeroRows                    ▷ we can do this trivialy by a for cycle
end procedure
```

construction of the dual. We can pick any arbitrary orientation for this, for example in the testing algorithm we picked an orientation from the vertex with the lesser index to the one with the greater index for each edge. Later, during the construction of the dual, we just pick either clockwise or counterclockwise orientation of the surface. This orientation will determine the orientaion of the edges in the dual graph in relation to the original edges. We will also want the orientation of the edges in relation to the orientation of the same edges in the irreducible cycles. We'll need this relation to compute the flow, as the flow on the cycle should be equal to zero in a consistent orientation. We'll compare the orientation we picked to an orientation of each irreducible cycle and for each edge we store if it was the same or opposite. The pseudocode for this can be seen in algorithm 4.

Algorithm 4 Algorithm for finding the orientation of irreducible cycles in comparison to graphs orientation

```
for cycle in irreducibleCycles do
    array<pair<int, bool>> cycleOrientaion
    for Edge c in cycle do
        for Edge e in edges do
            if e.first=c.first and e.second=c.second then
                cycleOrientaion[c] ← true
            else if e.first=c.second and e.second=c.first then
                cycleOrientaion[c] ← false
            end if
        end for
    end for
    result.push ← cycleOrientaion
end for
```

Dual graph

The last thing we'll need is to transform the graph we have, that is the line graph of the original graph we started with, into its dual. It is important to note, we want to retain the order of edges in the faces when we turn them into vertices and the order of edges in the vertices when we turn them into faces. We also want to retain the orientation of all the edges, just rotated according to our chosen plane orientation.

The function will take the graph and its faces as arguments and return the dual graph, the edges of the dual in the same ordering as in the original graph and a map showing the orientation of each edge in the dual.

The algorithm will take an ordering of edges, the same one we used before, in the

face algorithm. For each of these edges we will then find the two faces that contain this edge create an corresponding edge from one face to the other and compute its new orientation. We will also add the edge to the dual graph in the same index as was its position in each of the faces, so the order of edges is retained. See algorithm 5

After we have gone through all the edges, we'll have the whole dual graph together with its orientation and edge ordering. We now have all we need to begin the computation using mixed linear programming.

Algorithm 5 Algorithm for finding the Graph

```
for edge e in edges do
    pair found ← (-1,-1)                          ▷ variable for the first instance of this e
    for face in faces do
        for edge f in face do
            if f = e and found = (-1,-1) then
                found ← (face, f, graph)
            else if f = e then
                dual[face][f] ← found.first
                dual[found.first][found.second] ← face
                bool orientation ← findDualOrientation(face, edge)      ▷ algorithm 6
            end if
        end for
    end for
end for
```

Algorithm 6 takes the graph, current edge, denoted u-v, and one of the faces that include this edge, F. The algorithm returns true, if the next edge incident to v after u-v in the original representation of the graph is one of the edges that form the bounding cycle of F.

If findDualOrientation() returns true and we want to retain the same orientaion as per the edge sequence in the original graph, then the new the dual edge will incoming to F. If it returns false, the new edge will be outgoing from F.

Algorithm 6 Algorithm for the new orientation in the dual graph, this orientation is based on the order of edges in the original graph

```
procedure findDualOrientation(face, edge, graph)
    size ← graph[second].size();
    for edge e in graph[edge.second] do
        if e.second = edge.first then
            nextEdge ← graph[edge.second][(e+1)%size]        ▷ the next edge in the
second vertex after our edge
            for edge f in face do
                if f = nextEdge then
                    return true
                end if
            end forreturn false
        end if
    end for
end procedure
```

2.3 Calculating flow using mixed linear programming

For computing the circular flow number of the graph we'll be using balanced valuation r-flows. Using this approach it is possible to use mixed linear programming solvers for circular flow as was first done by Lukoťka [15] in 2023.

Construction of the mixed linear program

We will now transform the problem of finding a balanced valuation r-flow into mixed linear problem, and later use a LP solver to calculate the flow number, which as proven above, is equivalent to the balanced valuation of the graph.

From the definition 8 of balanced valuation r-flows 3. and 2.,

$$b(v) = \sum_{e \in E^+(v)} \phi(e) - \sum_{e \in E^-(v)} \phi(e) \leq \frac{r-2}{r}\, deg(v)$$

$$b(v) \leq deg(v) - \frac{2deg(v)}{r}$$

For the encoding of b, as $b \in \mathbb{Z}$ and integers in mixed linear programming are lower bounded by 0, to retain the parity of b we encode $b(v)$ for given vertex or irreducible cycle v of G followingly. We define m_v for any v, as the maximum value that $b(v)$ can attain. We assume that $r \leq 6$, as proven in Seymour's 6-flow theorem [18]. Then based on the above formula: $m_v = deg(v) - \lceil \frac{2deg(v)}{6} \rceil$ We then add an integer variable x_v for every v with a range of $[0, m_v]$. The relation between b and x is the following: $b(v) = 2x_v - m_v$.

We encode the ϕ function followingly. Let t be the variable representing the limit of $\phi(e)$, this will be representing $\frac{r-2}{r}$. Next for every edge $e \in E(G)$ we'll have a variable x_e that will represent the edges flow value. The LP solver will then, while minimizing t we have the following constraints:

1. $0 \leq x_v \leq m_v$, for every vertex or irreducible cycle v,
2. $-t \leq x_e \leq t$, for every edge e,
3. $\sum_{e \in E^+(v)} \phi(e) - \sum_{e \in E^-(v)} \phi(e) = 2x_v - m_v$, for every vertex or irreducible cycle v,

Where with the first formula we bound x_v to the range specified above. With the second formula, we satisfy the 2. condition of the balanced valuation r-flow definition, that is definition 8, where we define the range for the edge flow values. With the third set of constraints we will satisfy the 3. condition in definition 8. In these constraints we represent $b(v)$, using its assingned x_v, and m_v variables as was specified above.

The result we get from the solver, will be the minimized t, from which we'll get our result we'll calculate r as per following: $r = \frac{2}{1-t}$.

Chapter 3

Results

In this chapter we'll take a look at some results of using the algorithm and the linear programming solver to compute the circular flow indexes of several different cubic graphs. We'll also take a look at the comparison between the speed of the algorithm and the trivial algorithm using linear programming described in 2.1. For solving the mixed linear part of the algorithm we used lp_solve5.5 [1]. We ran these tests on a computer with Intel Core i5-10300H CPU @ 2.50GHz processor.

We tested this algorithm on some smaller graphs first, for testing bugcatching purposes adn as interesting comparison of the difference in time between the c++ part of the algoritm and the solver. For small orders of graphs the polynomialy complex algorithm could have slower times than the trivial algorithm. Namely, we used the K_4 graph and petersen graph.

As the main interest of this paper are the circular indexes of Snarks, we run this algorithm on the main major families of snarks, Flower Snarks, Goldberg snarks and Blanuša snarks. In 1.1 we mentioned Snarks with known chromatic indexes, we used these Snarks for result verification.

In table 3.1 we can see the results and time comparisons for our algorithms on flower

Flower Snarks				
index	vertex order	ϕ_c	time dual (s)	time trivial (s)
3	12	$\frac{7}{2}$	0,815	3,939
4	16	3	2,238	0,573
5	20	$\frac{17}{5}$	198,262	1253,579
7	28	$\frac{10}{3}$	31 052,234	47 654,323

Table 3.1: Running time of our algorithm (left) and the trivial algorithm(right) for flower snarks.

Small cubic graphs				
graph	vertex order	ϕ_c	time dual (s)	time trivial (s)
K4	4	3	0,001	0,001
Cubical graph	8	3	0,002	0,002
Pentagonal prism	10	3	0,004	0,005
Small non-planr graph	10	3	0,028	0,005
Petersen	10	$\frac{11}{3}$	0,329	3,246

Table 3.2: Running time of our algorithm (left) and the trivial algorithm(right) for some cubic graphs of small orders.

snarks. As we can see, the dual algorithm preforms better on small order snarks, J_3 and J_5. Flower snark J_4 isn't really a snark, as its chromatic index is 3, as is its circular index. We can see the trivial algorithm, preforms significantly better on J_4 then the dual algorithm. This may be caused by the relative speed of the lp solver compared to the rest of the algorithm when it is easy to find the correct circular index. The J_7 snarks runs faster on the dual then on the trivial algorithm, but not as remarkably. This is propbably caused by higher order graphs having more non-planar edges, thus having more irreducible cycles which cause the linear program to have more variables thus increasing complexity. The trivial program does not have these variables as it doesn't use the duality.

In table 3.2 we've taken a look at some small cubic graphs, mostly non-snarks, to compare the results. For the planar graphs the speed is mostly the same. The non-planar graph runs longer on the dual algorithm, this result may be caused by the added complexity in the c++ part of the code or it may be the additional variables and constraints in the linear program. This suggests that the less planar, that is more crossing edges and irreducible cycles there are, the slower the linear program runs. For more complex graphs, especially snarks as we can see for the Petersen graph, the dual algorithm gives us larger time savings.

In table 3.3 we've run the tests on the first and second Blanuša snarks and on the Goldberg snark G_3. We can see that, similarly to Flower snarks, the dual algorithm's runtime is significantly faster then the trivial algorithm.

Blanuša and Goldberg snarks				
graph	vertex order	ϕ_c	time dual (s)	time trivial (s)
First Blanuša snark	18	$\frac{10}{3}$	16,371	287,863
second Blanuša snark	18	$\frac{13}{4}$	27,108	96,586
Goldberg snark G_3	24	$\frac{10}{3}$	109,366	2873,073

Table 3.3: Running time of our algorithm (left) and the trivial algorithm(right) for Goldberg and Blanuša snarks.

Conclusion

In this thesis we looked at circular chromatic edge colorings, circular flows and the relationship between these two concepts . We used the concept of r-tensions to represent the duality of circular colorings. We have also mentioned several cubic graphs with already known circular edge colorings that were proven before and we have taken a look at some already existing algorithms for circular colorings as well as circular flows.

We have implemented an algorithm, which takes the graph, or its line, transforms it into its dual, while retaining all properties that we find important, the sequence of edges in the dual corresponds to the sequence in the faces of the graph and the orientation of edges in the dual is consistent with the original graph.

We have then implemented a mixed linear program that will using the arguments given by the previous algorithm compute the r-tension of the graph, from which we can get, due to the duality, the circular chromatic index of the original graph. To encode the r-tensions as a mixed linear program, we used the concept of balanced valuation r-flows, which are equivalent to a graphs circular r-flow ,slightly modified to include irreducible cycles in the graph, that are necessery to compute the circular chromatic number (or index) of a graph embedded in an orientable space other then plane, using this method.

We have also implemented a trivial algorithm that uses mixed linear programming to compute a graphs circular chromatic index without the use of the before mentioned duality. In the last part of this thesis we compared the results of these two algorithm on several snarks as well as some cubic graphs of low orders. We have found that the dual algorithm performed many times better on graphs of higher orders and about the same on graphs of low orders, most probably due to the polynomial complexity of the algorithm as opposed to the non-polynomial complexity of mixed linear programming solvers, that preform significantly worse on graphs of higher orders, so by making this part of the algorithm more effective, we have achieved faster runtimes.

The results from this algortihm, or algorithms of similar concept, could be used to more effectivly compute circular chromatic indexes of higher snarks. Further improvements could be made by optimizing the choice of the spanning tree in order for the linear program to have less variables or constraints.

Bibliography

[1] Michel Berkelaar, Kjell Eikland, and Peter Notebaert. lp_solve 5.5.2.11. website, 2020. `https://lpsolve.sourceforge.net/5.5/`.

[2] Matt DeVos, Luis Goddyn, Bojan Mohar, Dirk Vertigan, and Xuding Zhu. Coloring-flow duality of embedded graphs. *Transactions of the American Mathematical Society*, 357(10):3993–4016, 2004.

[3] Reinhard Diestel. *Graph theory*. Springer, 2000.

[4] Genghua Fan. Circular chromatic number and mycielski graphs. *Combinatorica*, 24:127–135, 2004.

[5] M. Ghebleh, D. Král', S. Norine, and R. Thomas. The circular chromatic index of flower snarks. *The Electronic Journal of Combinatorics*, 13, 2006.

[6] Mohammad Ghebleh. The circular chromatic index of goldberg snarks. *Discrete Mathematics*, 307(24):3220–3225, 2007.

[7] Jan Goedgebeur, Davide Mattiolo, and Giuseppe Mazzuoccolo. Computational results and new bounds for the circular flow number of snarks. *Discrete Mathematics*, 343(10):112026, Oct 2020.

[8] Jonathan L Gross, Jay Yellen, and Mark Anderson. *Graph theory and its applications*. Chapman and Hall/CRC, 2018.

[9] F Jaeger. Balanced valuations and flows in multigraphs. *Proceedings of the American Mathematical Society*, 55(1):237–242, 1976.

[10] Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972.

[11] William Klostermeyer and Cun Quan Zhang. (2 + e)-coloring of planar graphs with large odd-girth. *Journal of Graph Theory*, 33(2):109–119, 2000.

[12] Daniel Král', Edita Máčajová, Ján Mazák, and Jean-Sébastien Sereni. Circular edge-colorings of cubic graphs with girth six. *Journal of Combinatorial Theory, Series B*, 100(4):351–358, May 2010.

[13] Olívia Kunertová. Circular chromatic index of small snarks. Master's thesis, Univerzita Komenského v Bratislave Fakulta matematiky, fyziky a informatiky, 2017.

[14] Robert Lukot'ka. Determining the circular flow number of a cubic graph. *The Electronic Journal of Combinatorics*, pages P1–49, 2021.

[15] Robert Lukoťka. Balanced valuation flows. unpublished, 2023.

[16] Robert Lukoťka and Ján Mazák. Cubic graphs with given circular chromatic index. *SIAM Journal on Discrete Mathematics*, 24(3):1091–1103, 2010.

[17] Ján Mazák. Circular chromatic index of snarks. Master's thesis, Univerzita Komenského v Bratislave Fakulta matematiky, fyziky a informatiky, 2007.

[18] P.D Seymour. Nowhere-zero 6-flows. *Journal of Combinatorial Theory, Series B*, 30(2):130–135, 1981.

[19] A. Vince. Star chromatic number. *Journal of Graph Theory*, 12(4):551–559, Dec 1988.

[20] Wikipedia contributors. Graph embedding — Wikipedia, the free encyclopedia, 2024. [Online; accessed 24-May-2024].

[21] Xuding Zhu. Star chromatic numbers and products of graphs. *Journal of Graph Theory*, 16(6):557–569, 1992.

[22] Xuding Zhu. Circular chromatic number of planar graphs of large odd girth. *the electronic journal of combinatorics*, pages R25–R25, 2001.

[23] Xuding Zhu. Recent developments in circular colouring of graphs. In Martin Klazar, Jan Kratochvíl, Martin Loebl, Jiří Matoušek, Pavel Valtr, and Robin Thomas, editors, *Topics in Discrete Mathematics*, pages 497–550, 2006.