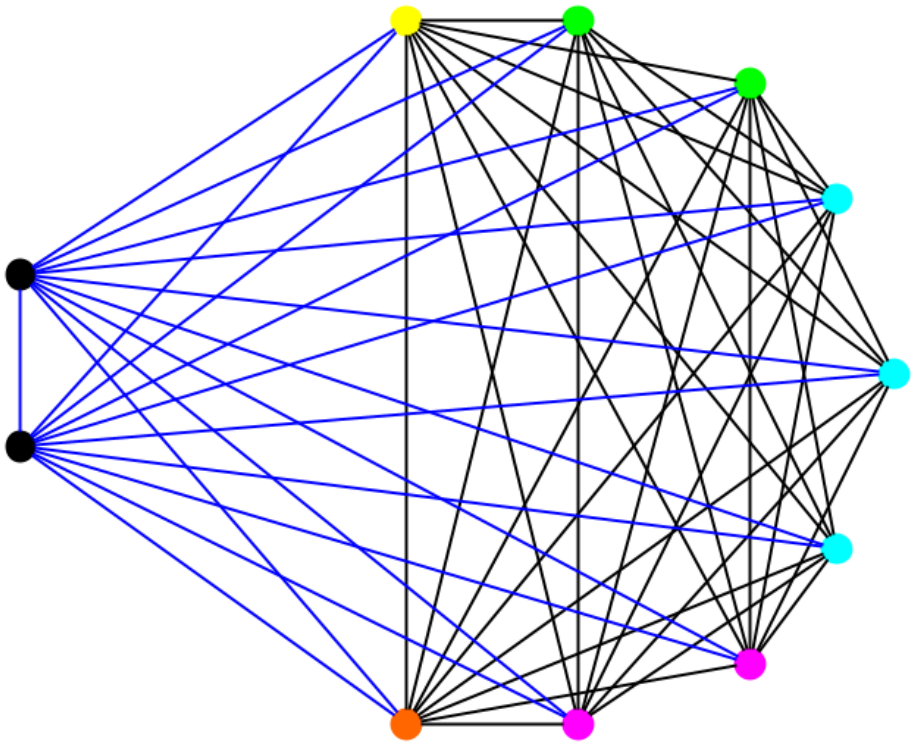
Two-Dimensional Nowhere-Zero Flows on Graphs

Determining Two-Dimensional Flow Numbers for Complete and Cubic Graphs

Anna Lusthof

Applied Mathematics
Delft University of Technology



Two-Dimensional Nowhere-Zero Flows on Graphs

Determining Two-Dimensional Flow Numbers for Complete and Cubic Graphs

by

A.V. Lusthof

to obtain the degree of Bachelor of Science
at the Delft University of Technology,

Student number:	5832810	
Project duration:	September 1, 2025 – November 27, 2025	
Thesis committee:	Prof. dr. D.C. Gijswijt,	TU Delft, supervisor
	Dr. ir. E. Lorist	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Laymen's summary

Directed graphs form a way to model various things, such as traffic flows or electrical systems. They are collections of dots which we call vertices, in which there are arrows from certain dots to others. In the mentioned applications, a value has to be added to each arrow to model the flow through that arrow; an example is shown in Figure 1. In a flow, the arrows pointing into each vertex should have the same total value as the arrows pointing out of it; otherwise, the system is 'leaking'. The flow number of a graph is the smallest number k such that we can make a flow on the graph using only values between 1 and $k-1$. Instead of one number, we can also add a pair of two numbers to each arrow; we then get a two-dimensional nowhere-zero flow.

This thesis researches two-dimensional flow numbers on certain graphs. For example, we determine the two-dimensional flow number of complete multipartite graphs, which are graphs in which the vertices are split into some groups, and all dots in different groups have a line between them. Furthermore, we approximate two-dimensional flow numbers of certain graphs with optimization models.

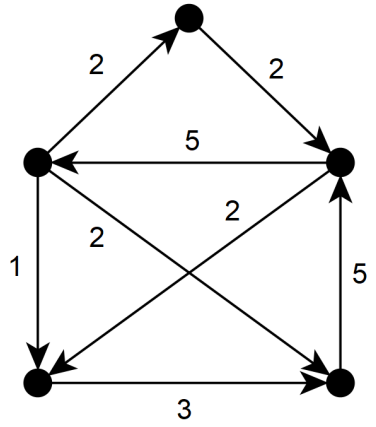


Figure 1: A circulation on a directed graph.

Abstract

A d-dimensional nowhere-zero flow of a graph $G = (V, E)$ is a function $f : \vec{E} \to \mathbb{R}^d$ that assigns a value to each directed edge such that for each $v_1v_2 \in E$, we have $f(v_1v_2) = -f(v_2v_1)$, each $v \in V$ has equal in- and outflow and $|f(e)| \geq 1$ for all $e \in \vec{E}$. If additionally $|f(e)| \leq r - 1$ for all $e \in \vec{E}$, we call this a d-dimensional nowhere-zero r-flow. The d-dimensional flow number of graph G is the smallest r such that a d-dimensional r-flow on G exists, and is denoted by $\phi_d(G)$. While the theory of nowhere-zero flows in either one or three or more dimensions is relatively well-developed, less is known about two-dimensional flow numbers. Therefore, this thesis aims to further research two-dimensional flow numbers.

First, an overview of existing conjectures and theorems about two-dimensional flow numbers is given, and some proofs are reviewed or rewritten. Then, the two-dimensional flow number is determined for all complete (multipartite) graphs, using some existing theorems as well as an induction proof. It is determined that the two-dimensional flow number of all complete (multipartite) graphs is equal to 2, except for K^4; it turns out that $\phi_2(K^4) = 1 + \sqrt{2}$.

An r-flow triangulation of a bridgeless cubic graph $G = (V, E)$ is a collection that represents each $v \in V$ as a triangle, where the three edges incident to v correspond to its triangle sides, such that all triangle sides have lengths from $[1, r - 1]$ and the two triangle sides corresponding to the same edge are attachable. Since G has an r-flow triangulation if and only if G has a nowhere-zero r-flow, triangulations can be used to determine upper bounds to $\phi_2(G)$. It is shown this way that for the Wagner graph M_8, it holds that $\phi_2(M_8) \leq 1 + 2\sin\left(\frac{2\pi}{9}\right)$. In addition, we aim to tackle the problem whether each cubic bipartite graph has a 'nice' 2-flow triangulation, where 'nice' means existing of one piece. This is shown for all Hamiltonian cubic bipartite graphs, and two non-Hamiltonian such graphs, but not for all cubic bipartite graphs.

Two-dimensional flow numbers can also be approximated computationally. A two-dimensional nowhere-zero $(t+1)$-flow is equivalent to a placement of points in the Euclidean plane such that each point lies between circles of radius 1 and t; also, some constraints have to be added to ensure each vertex has equal in- and outflow. Since this is a quadratic and not a linear optimization problem, determining the smallest such t requires extensive computational time. Still, results are found, including conjectures that state $\phi_2(M_8) = 1 + 2\sin\left(\frac{2\pi}{9}\right)$ and $\phi_2(T) = 1 + \sqrt{7/3}$, where T denotes Tietze's graph.

Contents

1

Introduction

Directed graphs are a powerful tool for modeling a wide range of real-world systems, from social networks to electrical circuits. However, a lesser-known application lies in a centuries-old mathematical problem: the four-colour problem. Originating from the nineteenth century, this problem posed a deceptively simple question: can every map be coloured using only four colours, such that no two neighboring regions share the same color? Over a hundred years later, the conjecture was finally proved by Appel and Haken in 1976, but their proof was controversial, since it relied on computer assistance [9].

Mathematically, the four-colour theorem is a statement about graph theory. A graph consists of vertices connected by edges, whereas a directed graph extends this idea by assigning a direction to each edge. This directionality allows us to model systems where the movement or relationship between elements matters. A nowhere-zero flow is a function that assigns values to the edges of a directed graph such that, at every vertex, the total "inflow" equals the total "outflow," and no edge is assigned a value of zero. An interesting result is that the four-colour theorem is equivalent to the statement that every bridgeless planar graph admits a nowhere-zero flow using values from the group $\mathbb{Z}/4\mathbb{Z}$. This gives an opportunity to find an alternative proof for the four-colour theorem.

The four-colour theorem has been a driving force behind the development of the theory behind nowhere-zero flows, which is also generalized to higher-dimensional flows. While the theory of nowhere-zero flows in three or more dimensions is relatively well-developed, the understanding of two-dimensional flows remains limited. Strong conjectures exist for higher dimensions, such as the S^2-flow conjecture, which posits that every bridgeless graph admits a d-dimensional nowhere-zero 2-flow for $d \geq 3$. In [8], many results for two-dimensional flows are presented, as well as a new approach to determining upper bounds of two-dimensional flow numers. In this thesis, the goal is to further research two-dimensional flow numbers for certain graphs, either by determining the exact two-dimensional flow number or by determining upper or lower bounds. Different methods will be used for this.

In Chapter 2, important concepts in the theory of directed graphs and flows are introduced and formally defined. In Chapter 3, an overview of the existing knowledge about multi-dimensional nowhere-zero flows is given. It contains many important theorems, for which some proofs are rewritten to make them more clear. In the later chapters, the theory introduced in Chapter 2 and Chapter 3 will be used to find (upper bounds to) two-dimensional flow numbers of certain graphs. In Chapter 4, the two-dimensional flow number of complete graphs will be determined. The same will be done for all complete multipartite graphs. Chapter 5 researches r-flow triangulations, a method suggested in [8] to determine upper bounds of two-dimensional flow numbers. Two main problems in this chapter are finding a flow triangulation of the Wagner graph, and finding nice flow triangulations for cubic bipartite graphs, which was an open problem from [8]. Lastly, Chapter 6 introduces a new method for approximating two-dimensional flow numbers. In this chapter, linear programming will be used to find two-dimensional nowhere-zero flows, for example on the Wagner graph and the Petersen graph. Furthermore, a general quadratic optimization model is posed that can determine upper bounds to or approximate two-dimensional flow numbers of graphs.

2

(Directed) graphs, circulations and flows

In this chapter, the necessary background information on the subject is given. In Section 2.1, the basics of graph theory are introduced, and graphs are defined. In Section 2.2, the theory of (nowhere-zero)-flows is introduced step-by-step. Multi-dimensional nowhere-zero flows and flow numbers, which are the main subject of this report, are also introduced in this section. In Section 2.3, connectivity is introduced, and it is explained why only bridgeless graphs are considered in this report. Section 2.4 gives a brief introduction to vertex- and edge-colourability, and introduces the concept of snarks.

2.1. Background information on graphs

First, we start with the basics by mathematically defining a graph. A *graph* is defined as a pair $G = (V, E)$, where V is the set of vertices and E is the set of edges between two vertices of V. Mathematically, $E \subseteq \binom{V}{2}$, where $\binom{V}{2}$ is the set of unordered pairs of vertices from V. The *order* of a graph $G = (V, E)$ is the number of vertices, that is, $|V|$. If an edge $e \in E$ has endpoints $v_1, v_2 \in V$, we also write $e = v_1 v_2$. Two vertices v_1 and v_2 are called *neighbours* if there exists an edge $e \in E$ such that $e = v_1 v_2$. Then we also say that e is *incident* to v_1 and v_2. Likewise, for edges, if there exists a vertex $v \in V$ such that both $e_1 \in E$ and $e_2 \in E$ are incident to v, then e_1 and e_2 are called *adjacent*. Furthermore, the *degree* $d(v)$ of a vertex v is the number of edges incident to v.

Now, some specific graph properties will be listed that are needed in further chapters. A *partition* of vertex set V is a set $\mathscr{P} = \{V_1, V_2, \cdots, V_n\}$ of nonempty sets that contain vertices, such that each vertex of G is in exactly one of the V_i's. Graph $G = (V, E)$ is *bipartite* if there exists a vertex partition $\mathscr{P} = \{V_1, V_2\}$ (of two sets) such that each $e \in E$ has one end in V_1, and the other in V_2. This concept can be generalized; a graph G is called *k-partite* if there exists a vertex partition $\mathscr{P} = \{V_1, V_2, \cdots, V_k\}$ such that each $e \in E$ has its two ends in different V_i.

Two interesting graph properties are containing an Euler or a Hamilton tour. First, a *walk* is a sequence $v_1 v_2 \cdots v_n$ of vertices such that for each i, the vertices v_i and v_{i-1} are neighbours. It is *closed* if the first and last vertex in the sequence are the same, so when the walk starts and ends at the same point. An *Euler tour* is a closed walk that crosses each edge in E exactly once. A graph G that has an Euler tour is called *Eulerian*. On the contrary, a *Hamilton tour* is a closed walk that crosses each vertex in V exactly once, and a graph containing such a walk is *Hamiltonian*.

Another special class of graphs is that of k-regular graphs. Graph $G = (V, E)$ is called *k-regular* if each $v \in V$ has degree k. In this report, a 3-regular graph will be called a *cubic* graph.

2.2. Directed graphs and flows

Now that we have defined graphs, we define directed graphs, to eventually define a nowhere-zero-flow, which is the subject of this report.

In Section 2.1, an edge was an element of the set of unordered pairs of vertices, denoted by $\binom{V}{2}$. Now, an *oriented edge* is an ordered pair of vertices, so oriented edges are in V^2. For example, a directed edge between vertices v_1 and v_2 is either v_1v_2 or v_2v_1, where e is directed from the first mentioned vertex to the second. This direction is also called the *orientation of e*. A *directed graph* is a graph $G = (V, E)$ with V the set of vertices and E a set of directed edges. All the orientations of the edges of G are together called an *orientation of G*.

Now, we can assign a value to each oriented edge, which will be referred to as a *flow value*. This does not have to be a real number; we can take values from any abelian group H. We define all these value assignments as a function $f : \vec{E} \mapsto H$, where $\vec{E}$ is the set of all oriented edges of the graph $G = (V, E)$. For the function to make sense in practical situations, we want it to suffice the following condition:

Condition 1. *For all oriented edges $e = v_1v_2 \in E$, it must hold that*

$$f(v_1v_2) = -f(v_2v_1).$$

In other words, if we assign the value $k \in H$ to an oriented edge $e = v_1v_2$, then the value assigned to the opposite orientation $e = v_2v_1$ has to be $-k$. Besides this, we also want the vertices v to satisfy *Kirchhoff's law*. First, for vertices $v \in V$, we define $\delta^{\text{in}}(v)$ as the set of edges oriented into v, and similarly, $\delta^{\text{out}}(v)$ as the set of edges oriented out of v. Kirchhoff's law states that for each $v \in V$, the inflow of v has to be equal to its outflow, where the *inflow* is equal to $\sum_{e\in\delta^{\text{in}}(v)} f(e)$, and the *outflow* to $\sum_{e\in\delta^{\text{out}}(v)} f(e)$. That gives us the second condition for function f:

Condition 2. *Each $v \in V$ has to satisfy Kirchhoff's law, that is, for each $v \in V$ it must hold that*

$$\sum_{e\in\delta^{\text{in}}(v)} f(e) = \sum_{e\in\delta^{\text{out}}(v)} f(e).$$

This leads to the following definitions, which are important in the rest of this report:

Definition 1.

*A **circulation** is a function $f : \vec{E} \to H$, from the set of all oriented edges to an abelian group H, that satisfies Conditions 1 and 2.*

*A **nowhere-zero flow** is a circulation $f : \vec{E} \to H$ where $f(e) \neq 0$ for all oriented edges $e \in \vec{E}$. Sometimes, this will just be called a flow.*

*A **nowhere-zero k-flow** is a nowhere-zero flow $f : \vec{E} \to \mathbb{Z}$ such that for all directed edges e, it holds that $0 < f(e) < k$.*

*The **flow number** of G is the smallest k such that a nowhere-zero k-flow on G exists.*

An example of a circulation that is also a nowhere-zero 6-flow is given in Figure 2.1.

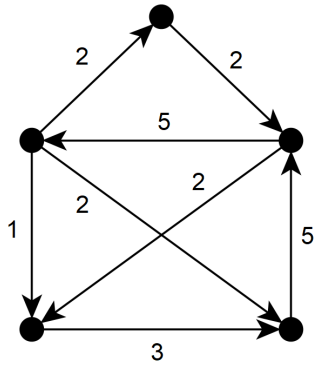


Figure 2.1: An example of a circulation with values from $\mathbb{Z}$, that is also a nowhere-zero 6-flow. One can check that it suffices Kirchhoff's law in all vertices.

Surpisingly, for abelian groups H, the number of nowhere-zero H-flows only depends on the order of H. In fact, for every graph G there exists a polynomial P such that for any finite abelian group H, the number of H-flows on G is $P(|H| - 1)$. This is proved by Tutte in 1954; a proof can be found in [3]. Of course, this implies the following corollary:

Corollary 1. *If H and H' are two finite abelian groups for which $|H| = |H'|$, then G has a nowhere-zero H-flow if and only if G has a nowhere-zero H'-flow.*

Furthermore, the following theorem gives an easy way to show whether a graph G has a nowhere-zero k-flow. Its proof can be found in [3].

Theorem 1. *G has a nowhere-zero k-flow if and only if G has a nowhere-zero $\mathbb{Z}/k\mathbb{Z}$-flow.*

The concept of flows can be extended to multiple dimensions. This is done in Definition 2.

Definition 2. *Let $r \geq 2$ be a real number and d a positive integer. A d-**dimensional nowhere-zero** r-**flow** on a graph $G = (V, E)$, also called an (r, d)-nowhere-zero flow, is an orientation of G, together with an assignment $f : \vec{E} \mapsto \mathbb{R}^d$, such that:*

1. *for all $e \in E$, the norm of $f(e)$ lies in the interval $[1, r-1]$;*
2. *for every vertex, the sum of the inflow and outflow is the zero vector in $\mathbb{R}^d$.*

*The infimum of the real numbers r such that G admits an (r, d)-nowhere-zero flow is called the d-**dimensional flow number** of G, denoted by $\phi_d(G)$.*

Note that the one-dimensional flow number from Definition 2 is not equivalent to the flow number from Definition 1; in the latter, flows are restricted to be integer, whereas in the case $d = 1$ of Definition 2, all real number greater than or equal to 1 are allowed. Also, note that 2 is the smallest possible flow number, independent of the dimension. Indeed, if a graph G has an (r, d)-nowhere-zero flow f with $r < 2$, then the norm of $f(e)$ lies in the interval $[1, r-1] = \emptyset$, which is impossible. If f is an $(2, d)$-nowhere-zero flow of G, then $\|f(e)\| = 1$ for all edges e. This means that all flow values are on the d-dimensional unit sphere, denoted by S^{d-1}. Therefore, for every graph G, the following lemma holds:

Lemma 1. *$\phi_d(G) = 2$ if and only if G has an S^{d-1}-flow.*

2.3. Connectivity

A graph $G = (V, E)$ is called *connected* if for every pair $v_1, v_2 \in V$, there exists a walk from v_1 to v_2, and *disconnected* if it is not connected. In simpeler words, a disconnected graph is one that is split up in multiple parts, while a connected graph exists of just one part. An example of a connected and a disconnected graph is drawn in Figure 2.2.

Figure 2.2: An example of a connected graph (left) and a disconnected graph (right).

In the context of flows and circulations, only connected graphs have to be considered. If a graph is disconnected, the flows on the seperate parts do not influence each other. Therefore, a flow on a disconnected graph is just a combination of flows on the connected subgraphs. Thus, from now on, connectivity is assumed.

Let $G = (V, E)$ be a connected graph. If G has a certain edge e such that if e is removed, G becomes disconnected, then e is called a *bridge*. It connects two disjoint vertex sets V_1 and V_2, that together form a partition of V. A graph without bridges is called *bridgeless*. Furthermore, a *cutvertex* is a vertex v such that the connected graph G becomes disconnected if v is removed (together with its incident edges). An example of a cutvertex and a bridge is given in Figure 2.3. For bridges, the following lemma holds.

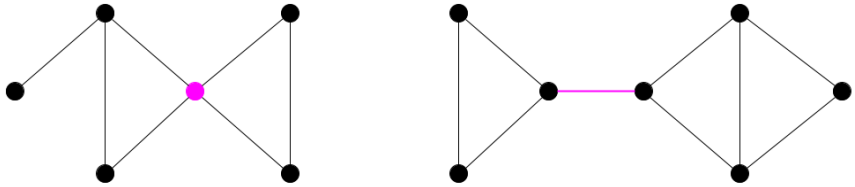


Figure 2.3: An example of a cutvertex (left) and a bridge (right).

Lemma 2. *If f is a circulation, and directed edge e is a bridge, then it must hold that $f(e) = 0$.*

Proof. For this proof we introduce some notation. First, for vertex sets $X, Y \subseteq V$, let $\vec{E}(X,Y)$ be the set of directed edges with one end in X and the other in Y. Let $f(X,Y) = \sum_{e \in \vec{E}(X,Y)} f(e)$. Now, by Condition 1, we know that $f(X,X) = \sum_{e \in \vec{E}(X,X)} f(e)$=0 since for each edge with both ends in X the flows in its opposite directions cancel each other. Furthermore, by Condition 2, $f(X,V) = \sum_{x \in X} f(x,V) = \sum_{x \in X} 0 = 0$. Now, let G be a graph with bridge e_b. Let $\{V_1, V_2\}$ be the vertex partition such that e_b is the only edge between V_1 and V_2. Then

$$f(e) = f(V_1,V_2) = f(V_1,V) - f(V_1, V \setminus V_2) = f(V_1,V) - f(V_1,V_1) = 0 - 0 = 0.$$

□

Now, the following corollary follows directly from Lemma 2. Because of this, only bridgeless graphs will be considered in this report.

Corollary 2. *If graph G has a bridge, it has no nowhere-zero flows.*

Note that the proof Lemma 2 also proves the following, more general lemma.

Lemma 3. *Let $G = (V,E)$ be a graph, and let $V' \subseteq V$. Then the total flow into V' must be* 0.

Proof. The total flow into V' is denoted as $f(V \setminus V', V')$. Following similar arguments as in the proof of Lemma 2, we get

$$f(V \setminus V', V') = f(V, V') - f(V', V') = 0 - 0 = 0.$$

□

A graph G is called k-connected if it is still connected after removing any $k-1$ vertices (together with their incident edges). This also has an equivalent term for edges; G is called ℓ-edge-connected if it is still connected after removing any $\ell - 1$ edges. Note that 1-edge-connectivity is equivalent to 'regular' connectivity. Furthermore, 2-edge-connectivity is equivalent to bridgelessness. Thus, every graph considered in this report is 2-edge-connected.

2.4. Vertex- and edge-colourability

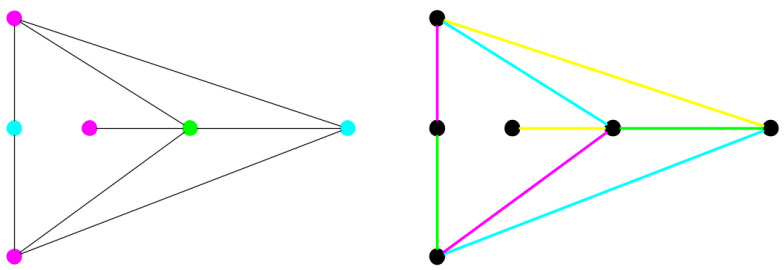


Figure 2.4: A vertex colouring (left) and edge colouring (right). This shows that this graph is 3-vertex-colourable and 4-edge-colourable.

A *vertex colouring* of a graph $G = (V, E)$ is a function $c : V \to C$ such that for any two neighbouring vertices v_1 and v_2, it holds that $c(v_1) \neq c(v_2)$. Here, C is the set of available colours. In simpeler words, it is a colouring of the vertices such that each two neighbouring vertices get different colours. A graph is called k-vertex-colourable if this is possible with k or less colours. Or, mathematically, if there exists a vertex colouring $c : V \to C$ for a colour set C with $|C| \leq k$. The *chromatic number* of graph G is the smallest k such that G is k-vertex-colourable. An edge colouring is the equivalent for edges; it is a function $c : E \to C$ such that for any two adjacent edges e_1 and e_2, it holds that $c(e_1) \neq c(e_2)$. Graph G is ℓ-edge-colourable if such a function exists for a colour set C with ℓ or less elements. Similar to the vertex-case, the *chromatic index* of a graph G is the smallest k such that G is k-edge-colourable. An example of a vertex and edge colouring of a graph is shown in Figure 2.4.

A *snark* is a bridgeless cubic graph that is not 3-edge-colourable. Their importance will become clear in Section 3.3; Theorem 13 gives a relatively low upper bound for all 3-edge-colourable cubic graphs, that is, all cubic graphs except snarks. The smallest snark is the Petersen graph, drawn in Figure 2.5. It has ten vertices.

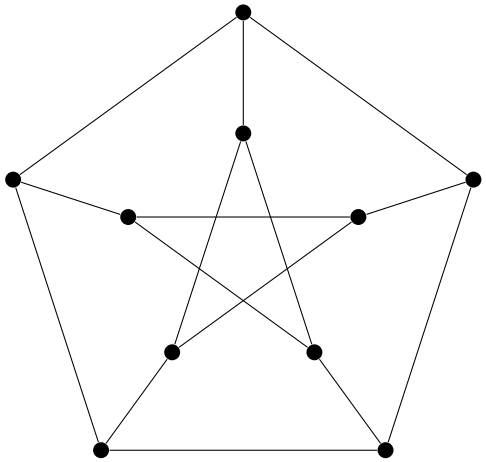


Figure 2.5: The Petersen graph, which is the smallest snark.

König's theorem states that for bipartite graphs, the chromatic number is equal to the maximum degree. A proof can be found in [3]. This implies the following lemma, which will be used in later chapters.

Lemma 4. *Let G be a cubic bipartite graph. Then G has chromatic index* 3.

This also implies that no snarks are bipartite.

2.5. Cycle covers

Contrary to the conventional definition of a cycle as a subgraph in which each vertex has degree 2, this paper uses a broader definition, which is better suited to this subject. In this report, a *cycle* is defined as a subgraph H of graph G in which each vertex has even degree. Furthermore, a *cycle double cover* of graph $G = (V, E)$ is a collection of cycles such that each $e \in E$ is in exactly two cycles. Lastly, we call a cycle $H = (V', E')$ in $G = (V, E)$ *induced* if there are no edges in $e = v_1 v_2 \in E$ such that $v_1, v_2 \in V'$, but $e \notin E'$.

Similar definitions also exist for directed graphs. Indeed, let G be a graph, and let $O(G)$ be the directed graph obtained after applying orientation O to G. a *directed cycle* is a subgraph H of $O(G)$ such that for each vertex v of H, the number of incoming edges of v equals its number of outgoing edges. Furthermore, oriented cycle double covers are defined in the following definition, from [8].

Definition 3. *The collection $\mathscr{C} = \{O_1(C_1), O_2(C_2), \cdots, O_k(C_k)\}$ of directed cycles of a graph G is called an **oriented cycle double cover** if every edge e of G belongs to exactly two cycles C_i and C_j and the directions of $O_i(C_i)$ and $O_j(C_j)$ are opposite on e. If $\mathscr{C}$ contains k cycles, we also call this an **oriented** k**-cycle double cover**.*

3

Overview of theorems and conjectures on multi-dimensional flow numbers

In [8], a collection of theorems and conjectures concerning multi-dimensional nowhere-zero flows is presented. This chapter aims to provide an overview of results on higher-dimensional nowhere-zero flows and their associated flow numbers. By summarizing the existing theory, we obtain the ability to construct new conjectures and theorems in subsequent chapters. This chapter is divided into results for two-dimensional flows (Section 3.2) and higher-dimensional flows (Section 3.1). Furthermore, Section 3.3 is about two-dimensional flow numbers of cubic graphs.

3.1. Three- or higher-dimensional nowhere-zero flows

In this Section, information on general d-dimensional nowhere-zero flows will be presented. Note that a d-dimensional nowhere-zero flow can always be extended to a $(d+n)$-dimensional nowhere-zero flow by extending the vectors of the d-dimensional flow with n zeros. The flow obtained by this is still a nowhere-zero flow. It is nowhere zero, since the first d components of the obtained vector in $\mathbb{R}^{d+n}$ cannot all be zero. Furthermore, it still satisfies Kirchhoff's law at every vertex; Kirchhoff's law already holds for the first d components, and for the other components, the sums of the flow into and out of a vertex are both zero.

Probably the most famous theorem about one-dimensional nowhere-zero flows is Seymours 6-flow theorem, which states that every bridgeless graph has a nowhere-zero 6-flow [10]. This can be extended to higher dimensions:

Theorem 2. *Let G be a bridgeless graph. Then $\phi_d(G) \leq 6$ for each dimension d.*

Proof. By Seymour's 6-flow theorem, $\phi_1(G) \leq 6$. The method of adding zeros gives the result $\phi_d(G) \leq 6$ for each dimension d. □

The 5-flow conjecture states that every bridgeless graph also has a nowhere-zero 5-flow [12]. This conjecture can also be generalized to multiple dimensions, leading to the following conjecture:

Conjecture 1. *Let G be a bridgeless graph. Then $\phi_d(G) \leq 5$ for all each dimension d.*

In this report, the notation S^n will be used for the $(n+1)$-dimensional unit sphere; S^1 is just the unit circle, and S^2 is the unit sphere in $\mathbb{R}^3$. Formally, $S^n := \{z \in \mathbb{R}^n : \|z\| = 1\}$. The paper [8] mentiones the S^2-flow conjecture, which states that each bridgeless graph G has a nowhere-zero flow with values from S^2. This would imply that $\phi_3(G) = 2$ for all bridgeless graphs G. Of course, this can again be extended to higher-dimensional flows, which leads to the following conjecture:

Conjecture 2. *Let G be a bridgeless graph and let $d \geq 3$. Then $\phi_d(G) = 2$.*

Observe that this conjecture is very strong: if proven true, it would render further research on d-dimensional flow numbers for $d \geq 3$ unnecessary. Unfortunately, the conjecture remains unproven at this time. The following theorem is a weaker version of Conjecture 2.

Theorem 3. $\phi_7(G) = 2$

Proof. In [2], it is proved that every bridgeless graph $G = (V, E)$ has seven cycles such that every edge of G is contained in exactly four of them. Using this result, we can construct a two-dimensional nowhere-zero flow such that the norm of each vector is exactly 1. Name the seven cycles $C_1, C_2, \cdots, C_7$. Since each vertex in a cycle has even degree, each cycle has an Euler tour. For each C_i, we define an orientation O_{C_i} of the cycle such that the Euler tour follows directions of the edges.

We give the whole graph $G = (V, E)$ an arbitrarily chosen orientation O_G. We define the function $f : E \to \mathbb{R}^7$. For each $e \in E$, we construct the vector $f(e)$ with seven components: $f(e) = (f_1(e), f_2(e), \cdots, f_7(e))$. The value of each $f_i(e)$ is determined using:

$$f_i(e) = \begin{cases} \frac{1}{2} & \text{if } e \in C_i \text{ and } O_G(e) = O_C(e). \\ 0 & \text{if } e \notin C_i \\ -\frac{1}{2} & \text{if } e \in C_i \text{ and } O_G(e) \neq O_C(e) \end{cases}$$

Since each edge is contained in exactly four cycles, the norm of the vector assigned to an edge will be

$$||f(e)|| = \sqrt{4 \cdot \left(\frac{1}{2}\right)^2} = 1.$$

Furthermore, $\sum_{e \in E(v)} f(e) = 0$ for all vertices v. This is because in an Euler tour, each vertex has the same number of incoming as outgoing edges, and we defined the values to be positive in the direction of the Euler tour. Function f has norm 1 everywhere, and satisfies Kirchoff's law, and therefore, f is a seven-dimensional nowhere-zero 2-flow. □

3.2. Two-dimensional nowhere-zero flows

Because of Conjecture 2, two-dimensional flow numbers are of particular interest. In [8], extensive research on two-dimensional nowhere-zero flows is presented. The following three theorems are proven results for general bridgeless graphs G:

Theorem 4. *Let G be a 6-edge-connected bridgeless graph. Then $\phi_2(G) = 2$.*

Theorem 5. *Let G be a bridgeless graph that admits an oriented k-cycle double cover for $k \in \{2, 3, 4, 5\}$. Then*

- $\phi_2(G) = 2$, *if* $k \leq 3$;
- $\phi_2(G) \leq 1 + \sqrt{2}$, *if* $k = 4$;
- $\phi_2(G) \leq \tau^2$, *if* $k = 5$,

where τ denotes the Golden Ratio $\frac{1+\sqrt{5}}{2}$.

Theorem 6. *Let G be a bridgeless graph. Then $\phi_2(G) \leq 1 + \sqrt{5}$.*

Additionally, the following fourth theorem holds. To state this theorem, we first introduce the definition of a *wheel graph*. It is a graph G consisting of a cycle C in which all vertices have degree 2, and a single vertex that is connected by an edge to each vertex in C. By W_n we mean the wheel graph that has a cycle of n vertices, together with one middle vertex; thus, W_n has $n + 1$ vertices. Two examples of wheel graphs are shown in Figure 3.1.

W_4 W_7

Figure 3.1: Two examples of wheel graphs: W_4 and W_7.

Theorem 7. *Let W_n be the wheel graph of order $n+1$, for $n \geq 3$. Then*

$$\phi_2(W_n) = \begin{cases} 2 & \text{if } n \text{ is even,} \\ 1+2\sin\left(\frac{\pi}{6} \cdot \frac{n}{n-1}\right) & \text{if } n \equiv 1,3 \mod 6, \\ 1+2\sin\left(\frac{\pi}{6} \cdot \frac{n+1}{n}\right) & \text{if } n \equiv 5 \mod 6. \end{cases}$$

Theorems 4 and 7 both have rather technical and complicated proofs. Theorem 4 is proved in [6], and Theorem 7 is proved in [7]. Theorems 5 and 6 are proved in [8]; these proofs are easier. Theorem 5 also implies the next corollary:

Corollary 3. *Let G be an Eulerian graph. Then $\phi_2(G) = 2$.*

Proof. Let $O(G)$ be the orientation of G that follows its Euler tour. Let $-O(G)$ be the orientation that follows its Euler tour backwards. These two form an oriented 2-cycle double cover of G, and by Theorem 5, it follows that $\phi_2(G) = 2$. □

Furthermore, the next theorem from [11] is very useful. However, the proof in [11] leaves certain steps unexplained. Therefore, another, independent proof is presented below.

Theorem 8. *If G has a nowhere-zero 3-flow, it has an S^1-flow.*

Proof. Assume that G has a nowhere-zero 3-flow. By Theorem 1, G also has a $\mathbb{Z}/3\mathbb{Z}$-flow. The goal is to prove that G has a nowhere-zero flow that uses only the complex solutions of the equation $w^3 = 1$. We call these three solutions 1, z and z^2, where $z = e^{2\pi i/3}$ and $z^2 = e^{4\pi i/3}$. In Figure 3.2, they are shown in the complex plane. It is clear that the sum of these three is 0. Moreover, it follows from the figure that if the sum of three complex numbers w_1, w_2, w_3 is 0 and one of them is a solution of the equation $x^3 = 1$, then the three numbers must be 1, z, and z^2.

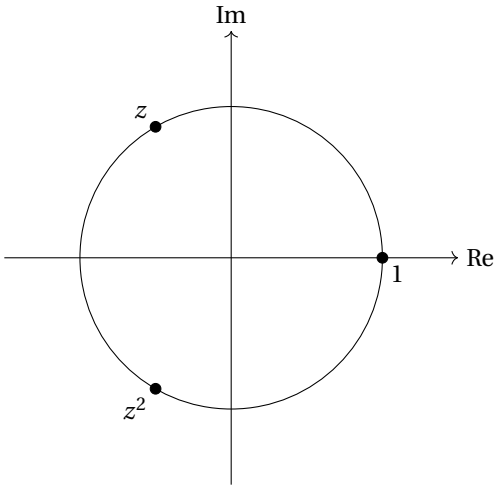


Figure 3.2: The three solutions of the equation $w^3 = 1$ drawn in the complex plane, on the unit circle.

Now, in the $\mathbb{Z}/3\mathbb{Z}$-flow on G, reverse the direction of each edge that has flow value 2. This way, all flow values are 1. If a vertex v has an incoming edge w_1v and outgoing edge vw_2 (both with flow value 1), we lift these two edges by replacing them with one edge w_1w_2. It is possible that there was already an edge between w_1 and w_2; in that case, we just add an extra edge, so we possibly get a multigraph. After this is repeated for all vertices, we end up with a (multi)graph in which all vertices either have only ingoing edges, or only outgoing. Because the nowhere-zero $\mathbb{Z}/3\mathbb{Z}$-flow is conserved by this operations, for each vertex, the number of incident edges must be a multiple of three. If vertex v has $k \cdot 3$ incident edges, we split vertex v into k vertices, which all get three of v's incident edges. Still, the nowhere-zero flow is conserved.

In the obtained graph $G' = (V', E')$, all vertices have either zero or three incident edges. Vertices with no incident edges are not relevant to the flow, so they will be ignored. If vertex $v' \in V'$ has incident edges, v' either has three ingoing or three outgoing edges. That means that the set $A \subseteq V'$ of vertices with three ingoing

edges and $B \subseteq V'$ of vertices with three outgoing edges form a vertex partition of G', since every edge in G' goes from a vertex in B to a vertex in A. Thus, G' is a cubic bipartite graph. Now, give each edge one of the three flow values $1, z, z^2$, in such a way that for each vertex, its three incident edges have three different flow values; this is possible since G is 3-edge-colourable by Lemma 4. Reverse the operations that created G' from G; this gives a nowhere-zero flow on G that uses the values 1, z and z^2. This is an S^1-flow on G. □

3.3. Cubic graphs

The reason to study cubic graphs is the following important theorem from [8]:

Theorem 9. *For all integers $d \geq 1$ and real numbers $r \geq 2$, the following two statements are equivalent:*

1. *Every bridgeless graph has a d-dimensional nowhere-zero r-flow*
2. *Every bridgeless cubic graph has a d-dimensional nowhere-zero r-flow*

In other words, an upper bound of the two-dimensional flow number of all (bridgeless) cubic graphs would also be an upper bound of the two-dimensional flow number of all (bridgeless) graphs. This makes the study of cubic graphs very interesting. The proof of this theorem is based on the proof provided in [5] for regular nowhere-zero flows, which also works for multi-dimensional flows.

Proof. The second statement is a special case of the first, so we only have to show that the second statement implies the first. Assume that every bridgeless cubic graph has a d-dimensional nowhere-zero r-flow. Let G be an arbitrary bridgeless graph that is not cubic. Without loss of generality, we can assume G is connected. We will construct a cubic graph G' from G by seperating each vertex with degree $n \neq 3$ into a cycle of n vertices. In Figure 3.3, an example of this is shown for a vertex with five edges.

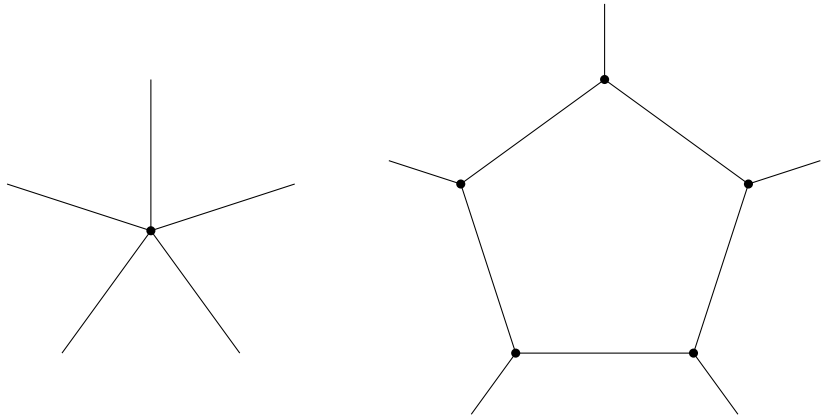


Figure 3.3: If a vertex has five incident edges, we split in into a cycle of five vertices. This way, each vertex gets degree 3.

Since G' is cubic, it has a d-dimensional nowhere-zero r-flow f' by our assumption. We use this f' to construct a flow f on G. Graph G is made from G' by contracting every obtained cycle of G' into one vertex. Since the sum of all flow values into a cycle is 0 by Lemma 3, the flow properties are conserved after contracting the cycles. Furthermore, all flow values of f were also values of f', so f is also a d-dimensional nowhere-zero r-flow. □

Now that we know why cubic graphs are so interesting, the rest of this section will be used to present theorems about two-dimensional flow numbers of cubic graphs, together with proofs. The presented theorems and proofs are from [8], unless stated otherwise.

Theorem 10. *Let G be a bridgeless cubic graph. Then $\phi_2(G) = 2$ if and only if G is bipartite.*

Proof. This proof is based on the proof in [11]. First, we will show that if $G = (V, E)$ is a bipartite cubic graph, G has an S^1-flow. Let (V_1, V_2) be the vertex partition of G such that each edge has one end in V_1 and the other in V_2. We define the direction of each $e \in E$ in such a way that the head of e is in V_1 and the tail of e in V_2. Let 1, z, and z^2 be the three complex solutions of the equation $w^3 = 1$, introduced in the proof of Theorem

8. Give each edge one of these three flow values, in such a way that for each vertex, its three indicent edges have three different flow values. This is possible, since G is 3-edge-colourable, again by Lemma 4. We obtain an S^1-flow on G.

It is left to prove that if $G = (V, E)$ is a cubic graph such that $\phi_2(G) = 2$, then G must be bipartite. If $\phi_2(G) = 2$, then G has an S^1-flow by Lemma 1. Let $v \in V$ be arbitrary. We can adjust the flow values by 'turning the circle', that is, multiplying all flow numbers by the same complex constant, in such a way that one of the three flow values of the incident edges of v is 1. Then, the other two edges must have values in $\{z, z^2, -z, -z^2\}$, depending on the direction of these edges. These three edges have another vertex at its other end, for which it also holds that the two other incident edges must have values in $\{1, -1, z, -z, z^2, -z^2\}$. Since G is finite and connected, this way we obtain a flow on G with values from $\{1, -1, z, -z, z^2, -z^2\}$. Reversing the direction of every edge that has flow value -1, $-z$ or $-z^2$ makes a flow that only uses values from $\{1, z, z^2\}$. Note that all vertices either have indegree or outdegree 0; otherwise, the sum of the flow in that vertex is not equal to 0. Let $\{V_1, V_2\}$ be a partition of the vertices, where V_1 contains all vertices with indegree 0 and V_2 contains all with outdegree 0. Then every edge has one end in V_1 and one in V_2. We can conclude that G is bipartite. □

Theorem 11. *Let G be a cubic bridgeless graph that contains a chordless cycle of length k. Let W_k be the wheel graph of order $n+1$. Then $\phi_2(G) \geq \phi_2(W_k)$.*

Proof. Let G be a graph that contains a chordless cycle C of length k. Assume that $\phi_2(G) < \phi_2(W_k)$. This implies that G has a two-dimensional nowhere-zero r-flow φ for an $r < \phi_2(W_k)$. Now, we contract all vertices of G that are not in C to one vertex. An example of this, for the cube graph, is given in Figure 3.4. The obtained graph is the wheel graph W_k. Furthermore, φ induces a flow μ on W_k, where each flow value of μ is also a flow value of φ. Thus, μ is a two-dimensional nowhere-zero r'-flow with $r' \leq r < \phi_2(W_k)$ on W_k, which is a contradiction. □

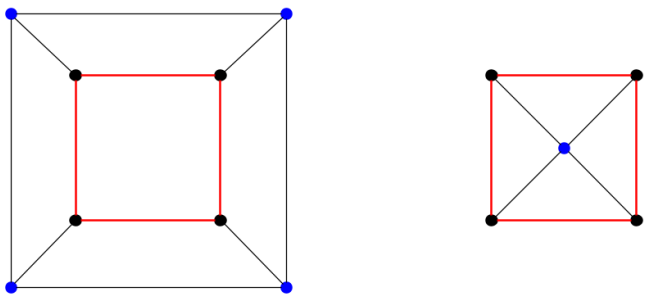


Figure 3.4: Left: an example of a graph with a chordless cycle, where the cycle is drawn in red. Right: the graph after contracting all vertices that are not in the cycle to one vertex.

Theorem 12. *Let n be odd and let P_n be the prism graph of order $2n$. Then $\phi_2(P_n) = \phi_2(W_n)$.*

Proof. Let n be an odd positive integer. Then P_n has an induced cycle of length n. From Theorem 11, it follows that $\phi_2(P_n) \geq \phi_2(W_n)$. Thus, it suffices to prove that $\phi_2(W_n) \geq \phi_2(P_n)$. Let φ be a two-dimensional nowhere-zero r-flow on W_n. We will prove that this also induces a two-dimensional nowhere-zero flow φ' on $\phi_2(P_n)$. Let $g_1, g_2, \cdots, g_n$ be the flow values of φ on the outer edges of W_n and let $f_1, f_2, \cdots, f_n$ be the flow values on the spokes of the wheel graph W_n. Now, for φ', give the outer edges the same direction and flow values g_i as the outer edges of W_n. For the edges connecting the inner and outer cycle in P_n, let the directions and flow values of φ' be the same as in W_n. Give the edges of the inner cycle of P_n the opposite direction of the outer cycle edges, but the same flow values g_i. An example of this, for $n = 7$, is shown in Figure 3.5.

This defines flow φ' on P_n. Indeed, for all the outer vertices of P_n, the sum of the flow is the same as in the outer vertices in W_n, which is 0. Furthermore, the sum of the flow in the inner vertices of P_7 is -1 times the sum of the flow of an outer vertex of P_7, which equals $-1 \cdot 0 = 0$. Furthermore, φ' uses the same values as φ. We conclude that $\phi_2(W_n) \geq \phi_2(P_n)$. □

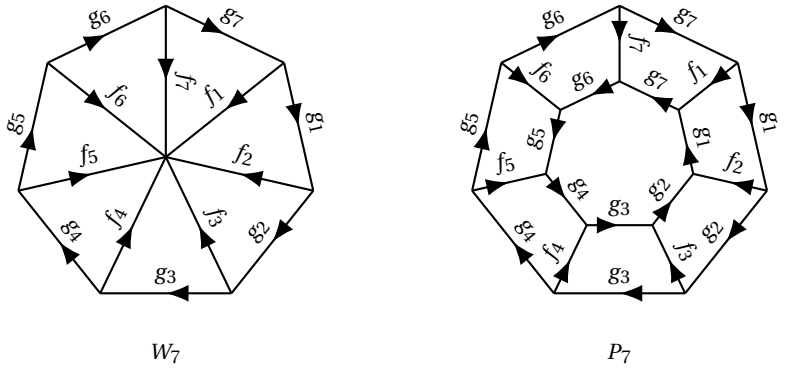


Figure 3.5: We create a flow on P_7, using a flow on W_7.

Theorem 13. *Let G be a 3-edge colourable cubic graph. Then $\phi_2(G) \leq 1+\sqrt{2}$.*

Proof. We show that every 3-edge-colourable graph $G = (V,E)$ has an oriented 4-cycle double cover. This proof is from [13]. First, we claim that G has a 3-cycle double cover. Indeed, define C_1 as the subgraph obtained by only using edges from E with colours 1 or 2. Subgraph C_2 is defined using edges with colours 1 and 3, and C_3 with 2 and 3. Since G is cubic, every vertex has exactly three incident edges, that are three different colours. Therefore, every vertex in each C_i has degree 2. Furthermore, each $e \in E$ is in exactly two of the three C_i's. This means that $\{C_1, C_2, C_3\}$ is a 3-cycle double cover of G.

Now, for each $i \in \{1,2,3\}$, let f_i be a circulation that is 1 on C_i and 0 elsewhere. We define the following circulations:

$$g_0 = \frac{f_1+f_2+f_3}{2}$$
$$g_1 = \frac{f_1-f_2-f_3}{2}$$
$$g_2 = \frac{-f_1+f_2-f_3}{2}$$
$$g_3 = \frac{-f_1-f_2+f_3}{2}$$

Let $e \in E$. One can check that there are exactly two $j \in \{1,2,3,4\}$ for which $g_j(e) \neq 0$. For these two j, call them j_1 and j_2, it holds that $g_{j_1}(e) = -g_{j_2}(e)$. For each $j \in \{1,2,3,4\}$, let D_j be a set of oriented edges e for which $g_j(e) = 1$. By these observations, the set $\{D_1, D_2, D_3, D_4\}$ forms an oriented 4-cycle double cover of G.

Theorem 5 states that for a graph G with an oriented 4-cycle double cover, it holds that $\phi_2(G) \leq 1+\sqrt{2}$. Thus, combining our result with this theorem finishes the proof. □

Theorem 14. *Let G be a 3-edge-colourable cubic graph that contains a triangle. Then $\phi_2(G) = 1+\sqrt{2}$.*

Proof. Let G be a 3-edge-colourable cubic graph with a triangle. By Theorem 13, we know that $1+\sqrt{2}$ is an upper bound for $\phi_2(G)$. Furthermore, a triangle is a chordless cycle of length 3. By Theorem 11 and Theorem 7, we know that

$$\phi_2(G) \geq \phi_2(W_3) = 1+2\sin\left(\tfrac{\pi}{6}\cdot\tfrac{3}{2}\right) = 1+2\sin\left(\tfrac{\pi}{4}\right) = 1+2\cdot\tfrac{1}{2}\sqrt{2} = 1+\sqrt{2}.$$

Since $1+\sqrt{2}$ is both an upper bound and a lower bound of $\phi_2(G)$, it must hold that $\phi_2(G) = 1+\sqrt{2}$. □

Theorem 15. *Let P be the Petersen graph. It holds that $\phi_2(P) \leq 1+\sqrt{7/3}$.*

Proof. This proof can be found in [8]; it will also be used in Chapter 5. □

Having established an overview of the existing knowledge on multi-dimensional flow numbers, we proceed to expand upon it in the next chapters.

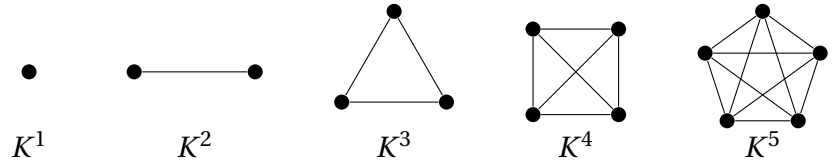
4

Complete graphs

The results of Chapter 3 allow us to determine the two-dimensional flow numbers for all complete graphs. In Section 4.1, we apply these theorems to complete graphs K^n, beginning with their formal introduction. Section 4.2 uses an induction proof to derive the two-dimensional flow number of all complete bipartite graphs. In Section 4.3, even more generalized results are presented by determining the two-dimensional flow number for all complete k-partite graphs.

4.1. Complete graphs

The complete graph on n vertices is the graph $K^n = (V, E)$ with n vertices such that for each pair $v_1, v_2 \in V$ of vertices, we have $v_1 v_2 \in E$. In other words, all the vertices are pairwise connected in a complete graph. For $n = 2, 3, 4, 5$, the complete graphs K^n are drawn below.



Using the theorems from Chapter 3, we can determine the two-dimensional flow numbers of complete graphs.

Corollary 4. *Let K^n be a complete graph.*

- *If $n = 3$ or $n \geq 5$, then $\phi_2(K^n) = 2$.*
- *If $n = 4$, then $\phi_2(K^n) = 1 + \sqrt{2}$.*

Note that if $n = 1$, the graph K^n has no edges. For $n = 2$, the complete graph K^n has only one edge, which is a bridge. That is why we only consider flow numbers of K^n for $n \geq 3$. Now, we proceed to prove the corollary.

Proof. If $n \geq 7$, the graph K^n is 6-edge-connected. By Theorem 4, this gives $\phi_2(K^n) = 2$. Furthermore, if n is odd, all vertices of K^n have even degree. Thus, K^n is Eulerian for odd n. By Corollary 3, this implies $\phi_2(K^n) = 2$ for odd n. This only leaves $\phi_2(K^4)$ and $\phi_2(K^6)$ undetermined. Since K^4 is a cubic graph with a triangle, $\phi_2(K^4) = 1 + \sqrt{2}$ by Theorem 14. An S^1-flow on K^6 can be constructed by combining S^1-flows on $K_{3,3}$ and triangles, as illustrated in Figure 4.1. Since the triangles are Eulerian and $K_{3,3}$ is a cubic bipartite graph, it admits an S^1-flow by Theorem 10. This completes the proof for all cases. □

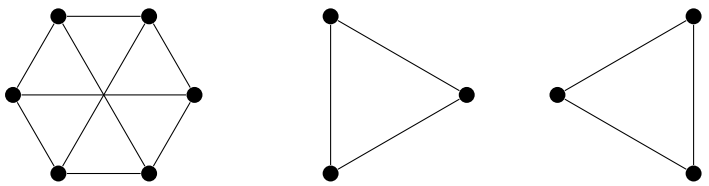
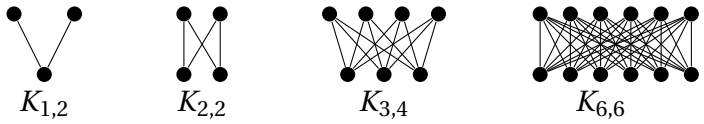


Figure 4.1: These graphs, $K_{3,3}$ and two triangles, all have an S^1-flow. Together, they form K^6.

4.2. Complete bipartite graphs

A complete bipartite graph $G = (V, E)$ is a bipartite graph with vertex partition (V_1, V_2), such that for each vertex pair (v_1, v_2) with $v_1 \in V_1$ and $v_2 \in V_2$, the edge $v_1 v_2 \in E$. We denote G as $K_{n,m}$ if $|V_1| = n$ and $|V_2| = m$. Note that $K_{n,m} = K_{m,n}$ for all n and m. Some examples of complete bipartite graphs $K_{n,m}$ for different n, m are drawn below.



The results presented in Section 4.1 naturally raise the question: can the two-dimensional flow number also be determined for complete bipartite graphs $K_{n,m}$? This is indeed possible:

Theorem 16. *Let $K_{n,m}$ be a complete bipartite graph with $n, m \geq 2$. Then $\phi_2(K_{n,m}) = 2$.*

Note that if n or m is equal to 1, the graph $K_{n,m}$ has a bridge. Therefore, $K_{1,m}$ has no possible nowhere-zero flows, which is why only the case $n, m \geq 2$ is considered.

Theorem 16 does not follow directly from the theorems in Section 3, unlike Corollary 4 for complete graphs K^n. The proof of theorem 16 is based on induction, and proves that every complete bipartite graph $K_{n,m}$ with $n, m \geq 2$ has an S^1-flow. By Theorem 1, this proves the theorem.

Proof. If $n \geq 2$ and $m \geq 2$ are even numbers, $K_{n,m}$ has an Euler tour, so $K_{n,m}$ has and S^1-flow by Corollary 3. Therefore, we only have to look at complete bipartite graphs $K_{n,m}$ where at least one of n or m is odd. First, we prove that $K_{2,m}$ has an S^1-flow for all $m \geq 2$, which we only have to prove for $m \geq 2$ odd. We prove this by induction. Note that $K_{2,3}$ has the oriented 3-cycle double cover shown in Figure 4.2:

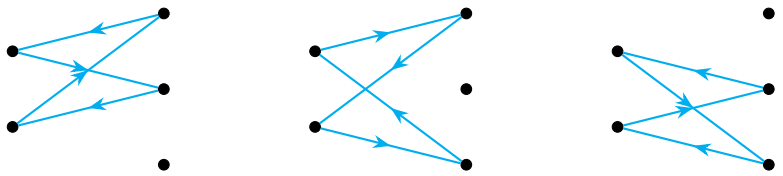


Figure 4.2: An oriented 3-cycle double cover of $K_{2,3}$.

It follows from Theorem 5 that $\phi_2(K_{2,3}) = 2$, which is equivalent to $K_{n,m}$ having an S^1-flow. Now that the base case is proven, we prove that if $K_{2,m}$ has an S^1-flow, then so does $K_{2,m+2}$. Let $m \geq 2$ be arbitrary. Assume that $K_{2,m}$ has an S^1-flow. Let $\{V_1, V_2\}$ be the partition of its vertex set, where $|V_1| = 2$ and $|V_2| = m$. Now, we add two nodes w_1, w_2 that both have two incident edges, with the two nodes $v_{1,1}, v_{1,2}$ of V_1 at its other ends. This gives us an Euler tour $v_{1,1}, w_1, v_{1,2}, w_2, v_{1,1}$ on the "new" part of the graph. An example of this, for adding two vertices to $K_{2,5}$, is given in Figure 4.3.

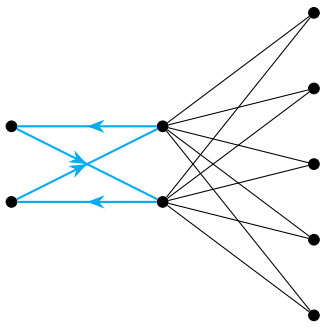


Figure 4.3: If $K_{2,5}$ has an S^1-flow, then an S^1-flow on $K_{2,7}$ can be obtained from combining flows on $K_{2,5}$ and $K_{2,2}$.

We assumed that $K_{2,m}$, which is the right or black part of Figure 4.3, has an S^1-flow. Since the "new" part of the graph has an Euler tour, shown in blue in the figure, it also has an S^1-flow by Corollary 3. This defines an

S^1-flow on the whole graph $K_{2,m+2}$. By induction, it is now proved that $K_{2,m}$ has an S^1-flow for all odd $m \geq 2$. Since this was also shown for all even $m \geq 2$, we conclude that $\phi_2(K_{2,m}) = 2$ for all $m \geq 2$.

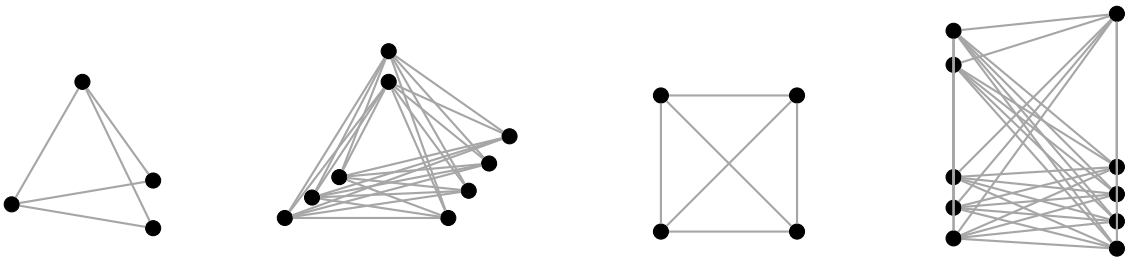
To prove a second base case for the general proof, we want to do the same for all $K_{3,m}$. We now know that $K_{3,2} = K_{2,3}$ has an S^1-flow. We can also use this information to prove that if $K_{3,m}$ has an S^1-flow, so does $K_{3,m+2}$; this follows the same argument as before. For the base cases of $K_{3,m}$, we also need to show that $K_{3,3}$ has an S^1-flow; luckily, this follows directly from Theorem 10. We can conclude that $K_{3,m}$ has an S^1-flow for all $m \geq 3$.

Of course, we want to prove that $K_{n,m}$ has an S^1-flow for all $n, m \geq 2$. We do that by fixing n and using induction on m. We proved that $K_{n,2} = K_{2,n}$ and $K_{n,3} = K_{3,n}$ both have S^1-flows, so the base cases hold. Again, let $\{V_1, V_2\}$ be the partition of $K_{n,m}$ where $|V_1| = n$ and $|V_2| = m$. For the induction step, we assume that $K_{n,m}$ has an S^1-flow and then add two vertices to V_2. Again, we get a "new" part of the graph, which is $K_{2,n}$; we indeed know that $K_{2,n}$ has an S^1-flow. We assumed the "old" part of the graph, $K_{n,m}$, had an S^1-flow as well, so $K_{n,m+2}$ has an S^1-flow defined by combining the "old" and "new" part. By induction, it is shown that for all $n, m \geq 2$, the complete graph $K_{n,m}$ has an S^1-flow, which is equivalent to $\phi_2(K_{n,m}) = 2$.

□

4.3. Complete k-partite graphs

The concept of complete bipartite graphs can also be generalized to complete k-partite graphs. These are graphs $G = (V, E)$ with a partition $V_1, V_2, \cdots, V_k$ such that for each vertex pair $v_i \in V_i, v_j \in V_j$ with $i \neq j$, we have $v_i v_j \in E$. If $|V_1| = n_1$, $|V_2| = n_2$, $\cdots$, $|V_k| = n_k$, we write G as $K_{n_1 n_2 \cdots n_k}$. Some examples of complete k-partite graphs are given below.



$K_{1,1,2}$ $K_{2,3,4}$ $K_{1,1,1,1}$ $K_{1,2,3,4}$

Note that $K_{1,1,\cdots,1}$ with n ones is just K^n. Therefore, its two-dimensional flow number is already established in Corollary 4. Theorem 16 can be generalized to the following result for k-partite graphs:

Proposition 1. *Let G be a complete k-partite graph, with partition $V_1, V_2, \cdots, V_k$. If $|V_i| \geq 2$ for all $i \in \{1, \cdots, k\}$, then $\phi_2(G) = 2$.*

Proof. Assume that G is a k-partite graph with partition $\{V_1, V_2, \cdots, V_k\}$ such that $|V_i| \geq 2$ for all $i \in \{1, \cdots, k\}$. We will prove that G has an S^1-flow, which is equivalent to $\phi_2(G)$ being equal to 2. Let $i, j \in \{1, \cdots, k\}$ be arbitrary. By Theorem 16, we know that there is an S^1-flow on the set of lines between V_i and V_j. By combining these flows on the sets of edges between all pairs V_i, V_j of sets in the partition, we obtain an S^1-flow on G. □

The constraint $|V_i| \geq 2$ for all $i \in \{1, \ldots, k\}$ in Proposition 1 is required to apply Theorem 16. However, for $k > 2$, nowhere-zero flows still exist on k-partite graphs with at least one set V_i in the partition satisfying $|V_i| = 1$. This raises the following question: What is the two-dimensional flow number for complete k-partite graphs with a partition $V_1, V_2, \ldots, V_k$ where at least one V_i has $|V_i| = 1$? By Corollary 4, we know that $\phi_2(K_{1,1,1,1}) = \phi_2(K^4) = 1 + \sqrt{2}$. Therefore, Proposition 1 does not hold for all k-partite graphs. However, there are some classes of k-partite graphs G for which we do know that $\phi_2(G) = 2$.

Proposition 2. *Let $K_{n,m,p}$ be a complete tripartite graph. Then $\phi_2(G) = 2$.*

Proof. If n, m and p are all at least 2, Proposition 1 shows that $\phi_2(K_{n,m,p}) = 2$. Assume $p = 1$. We know by Theorem 16 that for all n and m, the complete bipartite graph $K_{n+1,m+1}$ has an S^1-flow. Let $\{V_1, V_2\}$ be the vertex partition of $K_{n+1,m+1}$. Let $v_1 \in V_1$ and $v_2 \in V_2$, and contract v_1 and v_2 to one vertex v. An example of

this can be seen in Figure 4.4. The obtained graph is $K_{n,m,1}$, and the S^1-flow on $K_{n+1,m+1}$ induces an S^1-flow on $K_{n,m,1}$. Thus, $\phi_2(K_{n,m,p}) = 2$ for all $n, m, p \geq 1$. □

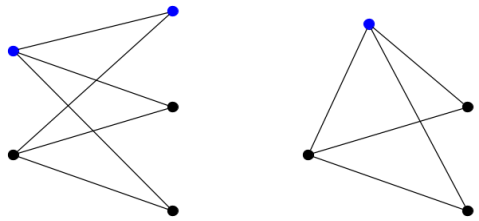


Figure 4.4: Contracting the two blue vertices from $K_{2,3}$ (left) results in the graph $K_{1,1,2}$ (right).

Proposition 3. *Let $n_1, \cdots, n_p \geq 2$ and $p \geq 1$. Let $K_{1,1,\cdots,1,n_1,n_2,\cdots,n_p}$ be a complete multipartite graph with vertex partition $V_1, \cdots, V_k$, where at least two sets in the partition have only one element. Then $\phi_2(K_{1,1,\cdots,1,n_1,n_2,\cdots,n_p}) = 2$.*

Proof. We prove this by induction on the number of sets in the vertex partition that contain only one element. First, we prove the base cases by showing that $K_{1,1,n_1,\cdots,n_p}$ and $K_{1,1,1,n_1,\cdots,n_p}$ both have an S^1-flow, and then we prove that we can add two sets of one vertex to the vertex partition, and still have an S^1-flow.

An S^1-flow on $K_{1,1,n_1,\cdots,n_p}$ can be constructed using S^1-flows on complete bipartite graphs and triangles. Indeed, for each $i, j \in \{1, \cdots, p\}$ with $i \neq j$, let $E_{i,j}$ be the set of all edges between V_{n_i} and V_{n_j}, where V_{n_i} is the set in the vertex partition for which $|V_{n_i}| = n_i$ (and the same for V_{n_j}). There is an S^1-flow on $E_{i,j}$, because these edges are exactly the edges of the complete bipartite graph K_{n_i,n_j}. The remaining edges are precisely the edges of $K_{1,1,\sum_i n_i}$, which has an S^1-flow by Proposition 2. An example of this is given in Figure 4.5. Combining all these flows gives an S^1-flow on $K_{1,1,n_1,\cdots,n_p}$.

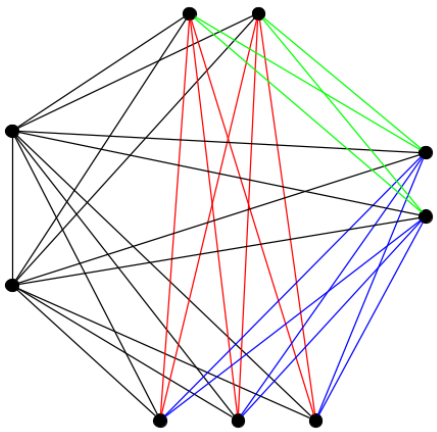


Figure 4.5: An S^1-flow on $K_{1,1,2,2,3}$. It consists of S^1-flows on complete bipartite graphs (red, green and blue) and an S^1-flow on $K_{1,1,7}$ (black).

An S^1-flow on $K_{1,1,1,n_1,\cdots,n_p}$ can be constructed similarly. We use the same flows on each $E_{i,j}$. Furthermore, there exists an S^1-flow on the triangle $K_{1,1,1} = K^3$. The remaining edges are precisely the edges of $K_{3,\sum_i n_i}$, which is a complete bipartite graph. On these edges, an S^1-flow exists because of Theorem 16. An example of this can be viewed in Figure 4.6. Here, the light blue lines are same lines as in K^3, the black lines are the same as in $K_{3,7}$, and the red, green and dark blue lines each are the same as in a certain complete bipartite graph.

For the sake of induction, from now on, the notation $K_{k\cdot 1,n_1,\cdots,n_p}$ will be introduced to denote $K_{1,1,\cdots,1,n_1,\cdots,n_p}$ with k sets of one vertex in the partition. We now assume that $K_{k\cdot 1,n_1,\cdots,n_p}$ has an S^1-flow. We want to show

that $K_{(k+2)\cdot 1,n_1,\cdots,n_p}$ also has an S^1-flow. Note that a flow on $K_{(k+2)\cdot 1,n_1,\cdots,n_p}$ can be obtained by combining a flow on $K_{k\cdot 1,n_1,\cdots,n_p}$ and $K_{1,1,\sum_i n_i+k\cdot 1}$. An example of this can be seen in Figure 4.7, where the two black nodes are vertices added in the induction step.

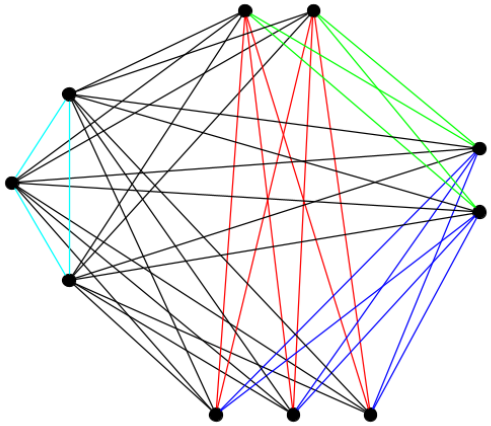


Figure 4.6: An S^1-flow on $K_{1,1,1,2,2,3}$ can be created by combining S^1-flows on the different coloured edge sets.

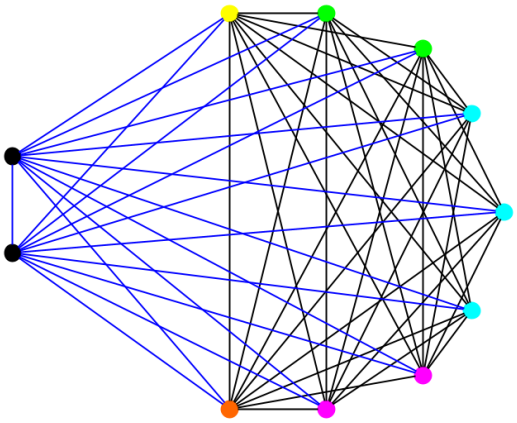


Figure 4.7: A flow on $K_{1,1,1,1,2,2,3}$ can be obtained by combining a flow on $K_{1,1,2,2,3}$ (black lines) and a flow on $K_{1,1,9}$ (blue lines).

Since we assumed that $K_{k\cdot 1,n_1,\cdots,n_p}$ has an S^1-flow, and $K_{1,1,\sum_i n_i+k\cdot 1}$ has an S^1-flow by Proposition 2, we can conclude that $K_{(k+2)\cdot 1,n_1,\cdots,n_p}$ has an S^1-flow too. □

Proposition 4. *Let G be a complete k-partite graph for some $k \geq 5$. Then $\phi_2(G) = 2$.*

Proof. The following three cases cover all possibilities:

- If the vertex partition of G contains only sets of one vertex, then G is a complete graph K^n for some $n \geq 5$. By Corollary 4, we then know that $\phi_2(G) = 2$.
- If the vertex partition of G contains at least two sets of one vertex, and also at least one set of more than one vertex, Proposition 3 shows $\phi_2(G) = 2$.

- If the vertex partition of G contains at most one set of one vertex, G is 6-edge-connected. Indeed, the smallest such G is $K_{1,2,2,2,2}$, which is 6-edge-connected; adding more vertices to this complete k-partite graphs only increases the connectivity. By Theorem 4, in this case $\phi_2(G) = 2$ holds too.

□

Theorem 16 and Propositions 2 and 4 together show that every complete k-partite graph has an S^1-flow, except if $k = 4$; this is the only case left. From now on, let G be a complete 4-partite graph. Propositions 1 and 3 say that if the vertex partition of G has zero, two or three sets of one element, $\phi_2(G) = 2$. If $G = K_{1,1,1,1} = K^4$, then $\phi_2(G) = 1 + \sqrt{2}$. Thus, only for $K_{1,n,m,p}$ with $n, m, p \neq 1$, the two-dimensional flow number is not yet determined. However, $K_{1,2,3,3}$ and all G with more vertices are 6-edge-connected, and therefore have $\phi_2(G) = 2$.

That only leaves two graphs: $K_{1,2,2,2}$ and $K_{1,2,2,3}$. The last challenge is to determine the two-dimensional flow number of these two graphs. By Theorem 8, it suffices to show that $K_{1,2,2,2}$ and $K_{1,2,2,3}$ both have a nowhere-zero 3-flow. By Theorem 1, this is equivalent to showing that they both have a $\mathbb{Z}/3\mathbb{Z}$-flow. Such flows indeed exist, which is shown in Figure 4.8. One can check that Kirchhoff's law is satisfied in each vertex.

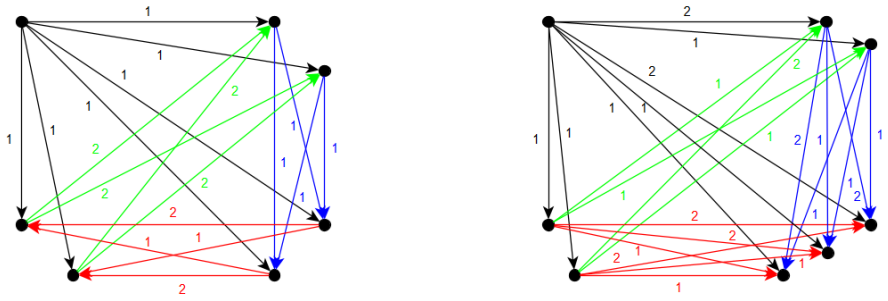


Figure 4.8: A $\mathbb{Z}/3\mathbb{Z}$-flow on $K_{1,2,2,2}$ (left) and $K_{1,2,2,3}$ (right). Kirchhoff's law is satisfied in each vertex, because in $\mathbb{Z}/3\mathbb{Z}$, calculations are modulo 3.

Now, the two-dimensional flow number for all complete k-partite graphs is finally determined. The results of this paragraph are summarized in the next theorem:

Theorem 17. *Let G be a complete k-partite graph. If $G = K^4$, then $\phi_2(G) = 1 + \sqrt{2}$. Else, $\phi_2(G) = 2$.*

5

r-flow triangulations

In [8], a new method to determine upper bounds of two-dimensional flow numbers is introduced, specifically for cubic graphs. The idea is to represent flows in a geometric way; a two-dimensional flow on a cubic graph is represented as a collection of triangles in the Euclidean plane, called an r-flow triangulation. In Section 5.1, r-flow triangulations are formally defined. In Section 5.2, the smallest cubic graph is determined for which the two-dimensional flow number is not yet known, and a flow triangulation is used to determine an upper bound to its two-dimensional flow number. Lastly, in Section 5.3, so-called "nice" flow triangulations are searched for cubic bipartite graphs.

5.1. Definition

To formulate a definition of r-flow triangulations, we first need the following definition of 'attachable', from [8]:

Definition 4. *Let s_1 and s_2 be sides of triangles T_1 and T_2, respectively. Sides s_1 and s_2 are called attachable if we can translate T_1 to a triangle T_1' in such a way that the image of s_1 coincides with s_2 and T_1' and T_2 have no common interior points.*

This means that attachable sides need to be parallel, of the same length and have their triangles on opposite sides. Using this definition, the geometric principle for representing two-dimensional flows on cubic graphs is formally defined in the following way in [8]:

Definition 5. *An r-**flow triangulation** of a bridgeless cubic graph G is a collection $\mathscr{T}$ containing a triangle T_v for each vertex v of G such that:*

1. *for each $v \in V(G)$, each edge incident to v corresponds to a unique side of T_v;*
2. *lengths of sides of all triangles from $\mathscr{T}$ are from the interval $[1, r-1]$;*
3. *for each edge $uv \in E(G)$, the sides of the triangles T_u and T_v corresponding to uv are attachable.*

Note that $r-1$ is actually the ratio between the largest and smallest triangle side. If the lengths of the sides in $\mathscr{T}$ are from an interval that does not start at 1, we can always multiply each side length with a constant such that the multiplied lengths are from the interval $[1, r-1]$, where $r-1$ is the ratio between the lengths of the longest and shortest sides in $\mathscr{T}$.

The goal of this section is to use r-flow triangulations to determine upper bounds of $\phi_2(G)$ for different cubic graphs G. In [8], the following theorem is proved:

Theorem 18. *Let G be a bridgeless cubic graph. Then G has an r-flow triangulation if and only if G has a two-dimensional r-flow.*

This implies that if we find an r-flow triangulation of a certain (bridgeless) cubic graph G, then $\phi_2(G) \leq r$. In [8], this principle is used to prove Theorem 15 from Chapter 3, which states that the two-dimensional flow number of the Petersen graph is at most $1+\sqrt{7/3}$.

5.2. r-flow triangulations of the Wagner graph and other Möbius ladders

In Section 3.3, it was proved that every bridgeless graph has a d-dimensional nowhere-zero r-flow if and only if every bridgeless cubic graph has a d-dimensional nowhere-zero r-flow (Proposition 9). Therefore, it is interesting to investigate the flow numbers of cubic graphs. Note that a cubic graph with an odd number of vertices does not exist. Indeed, let $V(G)$ be the set of vertices of a graph G. The number of edges in a cubic graph G is equal to $\frac{\#V(G)\cdot 3}{2}$, since each vertex is incident with three edges, and each edge has to be counted just once. If $\#V(G)$ is odd, this would mean G does not have an integer number of edges, which is a contradiction.

The first goal in this section is to find the smallest cubic graph for which the two-dimensional flow number is not yet known (spoiler alert: it is the Wagner graph). To do this, we investigate all cubic graphs of small order. First, if cubic graph G has only four vertices, it must be K^4. We know that $\phi_2(K^4) = 1+\sqrt{2}$ by Corollary 4. Furthermore, there are two cubic graphs with six vertices, drawn in Figure 5.1.

Figure 5.1: The only two cubic graphs with six vertices.

The smallest snark has ten vertices, so now we can still assume our graphs are 3-edge-colourable. Since the left graph in Figure 5.1 is 3-edge-colourable and has a triangle, its two-dimensional flow number must be $1+\sqrt{2}$ by Theorem 14. The right graph in the figure is the bipartite graph $K_{3,3}$, for which $\phi_2(K_{3,3}) = 2$ by Theorem 10. Thus, the smallest cubic graph with an unknown two-dimensional flow number has eight or more vertices. There are five bridgeless cubic graphs of order eight, shown in Figure 5.2. The first three graphs have a triangle, and are 3-edge-colourable, so for these it also holds that $\phi_2(G) = 1+\sqrt{2}$. The fourth graph is the cube graph, which is bipartite. That only leaves the fifth graph in the figure, which is also known as the Wagner graph. Three ways to draw it are shown in Figure 5.3.

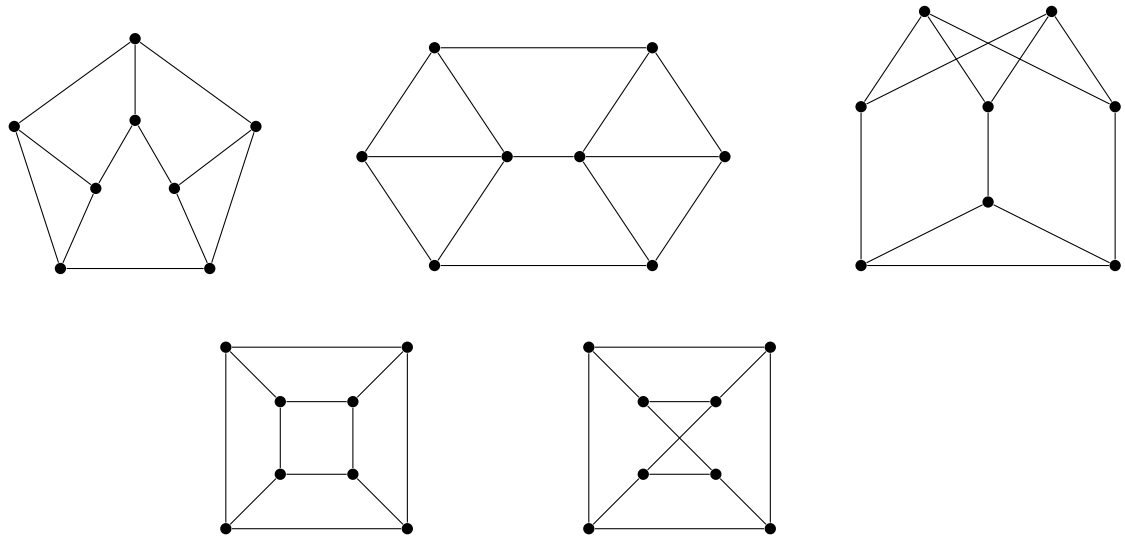


Figure 5.2: The five cubic graphs with eight vertices.

The Wagner graph is actually the Möbius ladder M_8. For even numbers n, the *Möbius ladder* M_n is defined as a graph with a cycle of n vertices, in which there is also an edge from each vertex to each opposite in the cycle. Note that Möbius ladders are cubic graphs, and that $M_4 = K^4$ and $M_6 = K_{3,3}$. They can be drawn in multiple ways, but the two most common are the middle and right image of Figure 5.3; the left drawing of the Wagner graph cannot be extended to larger Möbius ladders.

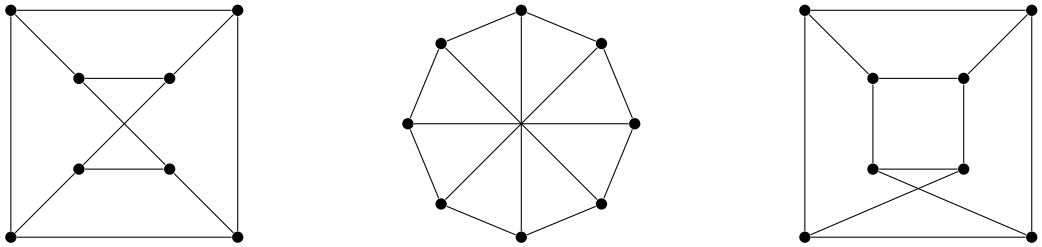


Figure 5.3: Three ways to draw the Wagner graph.

Note that the Wagner graph is neither bipartite nor does it contain a triangle. Therefore, we cannot use the theorems in Section 3.3 to determine the exact two-dimensional flow number of the Wagner graph. However, we can use Theorem 11 to determine a lower bound. Note that the Wagner graph contains a chordless cycle of length 5. Using the theorem, combined with Theorem 7, we can find a lower bound for the Wagner graph:

$$\phi_(M_8) \geq \phi_2(W_5) = 1 + 2\sin\left(\frac{\pi}{6} \cdot \frac{5+1}{5}\right) = 1 + 2\sin\left(\frac{\pi}{5}\right) \approx 2.1756$$

Furthermore, Theorem 13 gives an upper bound for the two-dimensional flow number. It states that for every cubic graph G that is 3-edge-colourable, $\phi_2(G) \leq 1 + \sqrt{2}$; therefore, $1 + 2\sin\left(\frac{\pi}{5}\right) \leq \phi_2(M_8) \leq 1 + \sqrt{2}$. The question is: can we find a better upper bound than $1 + \sqrt{2}$?

Here, we can use the theory of r-flow triangulations. In [8], the two flow triangulations in Figure 5.4 of the graphs K_4 and $K_{3,3}$ are presented. Here, the grey lines represent the graph, and the black lines represent the r-flow triangulations, where r equals $1 + \sqrt{2}$ for K_4 and 2 for $K_{3,3}$. Note that the graph is drawn in a different way than usual; the lines pointing outward are connected to their opposite lines.

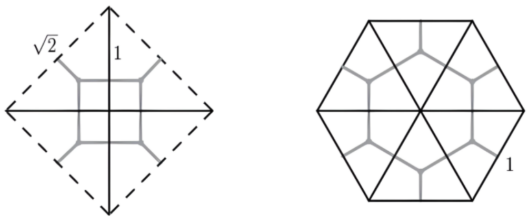


Figure 5.4: Optimal flow triangulations for K_4 and $K_{3,3}$. From [8].

Simirlarly, we can make an r-flow triangulation of the Wagner graph, which can be drawn in the same way as the graphs in Figure 5.4. This is shown in Figure 5.5.

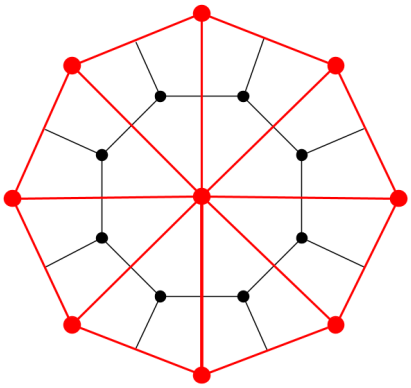


Figure 5.5: A flow triangulation of the Wagner graph. The graph is drawn in black, and the triangulation in red.

All triangles in this triangulation have one shorter side, and two equally long sides. If we set the length of the short side to 1, the length of the longer sides becomes $r-1 = \frac{1}{2}/\sin\left(\frac{1}{8}\pi\right) \approx 1.3066$. The result is that $\phi_2(G_W) \leq 2.3066$.

Of course, we can extend this to larger Möbius ladders. Applying the same triangulation method to M_{10} and M_{12} gives upper bounds of 2.6180 and 2.9318 respectively. All möbius ladders are 3-edge-colourable; to obtain a 3-edge-colouring, alternate colours 1 and 2 for the outer cycle edges, and assign colour 3 to the edges connecting the opposite vertices. Therefore, 2.6180 and 2.9318 are very weak upper bounds; we already knew that $\phi_2(G) \leq 1+\sqrt{2} \approx 2.4142$ for all M_n by Theorem 13. For even larger Möbius ladders, the ratio between the longest and shortest side of such a triangulation becomes even larger. Therefore, we must also explore alternative approaches to defining r-flow triangulations of Möbius ladders.

In [8], an r-flow triangulation is found for the Petersen graph. It is shown in Figure 5.6. In this picture, three five-cycles of the Petersen graph are drawn central; the other cycles are 'split open'. The Wagner graph also has cycles consisting of five vertices. This leads to the drawing of the Wagner graph shown in Figure 5.7.

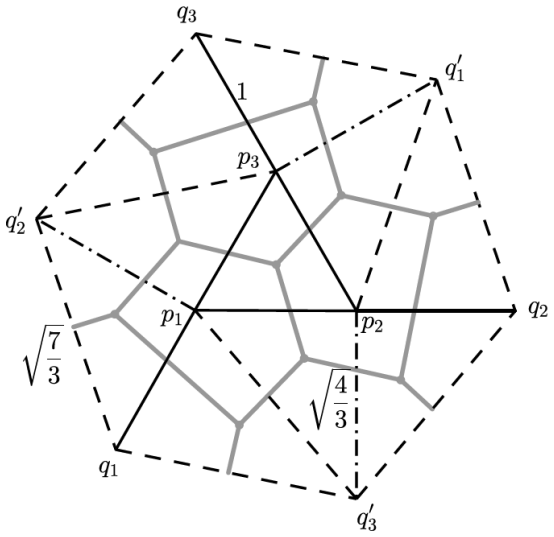


Figure 5.6: An r-flow triangulation of the Petersen graph. From [8].

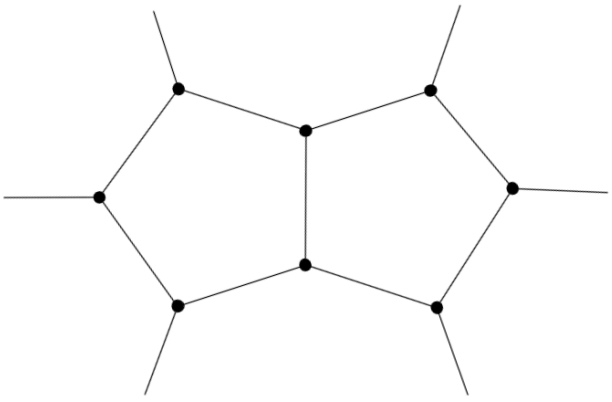


Figure 5.7: Another way to draw the Wagner graph, on which a flow triangulation can be drawn.

We can draw a triangulation on this Wagner graph in a natural way, shown in Figure 5.8.

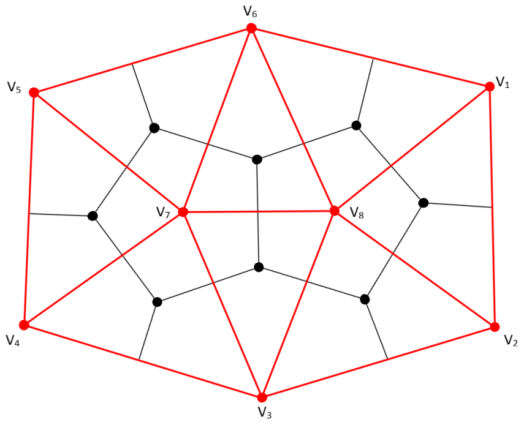


Figure 5.8: A flow triangulation on this Wagner graph. The graph is drawn in black, and the triangulation in red.

Of course, some triangle sides can be drawn longer or shorter than in the figure; the only restriction is that the sides corresponding to the same edge in the graph should be attachable. Therefore, the question is: how can we draw this triangulation in such a way that the ratio between the longest and shortest side is minimal? First, the shortest line should be v_7v_8, so we choose the length of v_7v_8 to be 1. Indeed, making this line longer only makes the whole figure wider, which is not ideal. Furthermore, note that the triangulation does not have to be symmetric along the horizontal vertical axis of Figure 5.8. Indeed, nodes v_3 and v_6 of the triangulation can be shifted to the right or left; the only restricting is that the sides that should be attachable, stay attachable. An example of an assymetrical flow triangulation of the Wagner graph is illustrated in Figure 5.9.

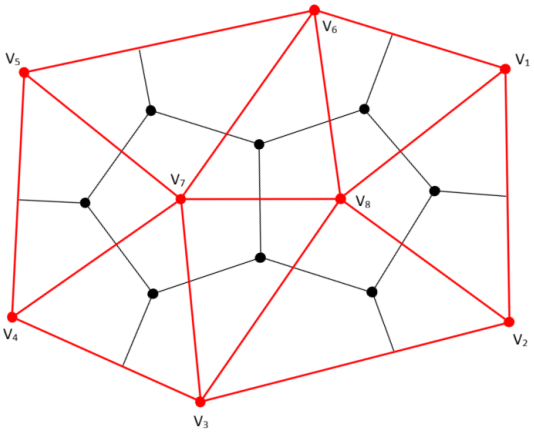


Figure 5.9: An asymmetrical flow triangulation of the Wagner graph.

However, this way, the lines v_5v_6 and v_2v_3 only become longer, while the lines v_1v_6 and v_2v_3 become shorter. Since the goal is to minimize the length of the longest side, this is not optimal. We can conclude that placing v_3 and v_6 in the middle such that v_4v_3, v_2v_3, v_5v_6 and v_1v_6 are of equal length is optimal.

The nodes v_3 and v_6 can also be placed closer to the graph. This also makes the lines v_1v_6, v_5v_6, v_2v_3 and v_4v_3 shorter. However, each edge length of the triangulation has to be greater than or equal to one, so the closest we can place v_6 and v_3 to the graph is such that v_6v_7, v_6v_8, v_3v_7 and v_3v_8 all get length 1. This is the optimal placement of v_6 and v_3. Denote the angle between the lines v_iv_j and v_jv_k as $\angle v_iv_jv_k$. Furthermore, denote the triangle with sides v_iv_j, v_jv_k and v_kv_i as $\triangle v_iv_kv_j$. The angles $\angle v_6v_7v_8$, $\angle v_6v_8v_7$, $\angle v_3v_7v_8$ and $\angle v_3v_8v_7$ are 60°, since triangles $\triangle v_6v_7v_8$ and $\triangle v_3v_7v_8$ are equilateral triangles. Now, it would be optimal to make the lines v_3v_4, v_4v_5 and v_5v_6 of equal length; the same for the right side. Also, note that we can just set the lengths of v_7v_4 and v_7v_5 to 1; that only makes other sides shorter, and none longer. In conclusion, the flow triangulation in Figure 5.10 is the optimal flow triangulation on this drawing of the Wag-

ner graph. This gives a maximal flow value of $2\sin(40°) \approx 1.285575$. Thus, we have found the upper bound $\phi_2(M_6) \leq 1 + 2\sin 40° \approx 2.285575$.

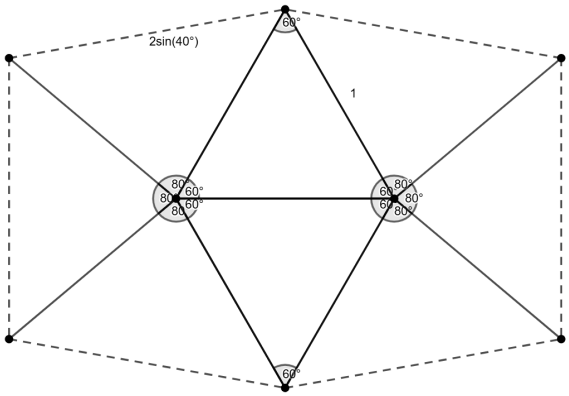


Figure 5.10: The optimal flow triangulation of the Wagner graph. All solid lines have length 1, and all dashed lines have length $2\sin(40°) \approx 1.285575$.

We can also attempt to generalize this to larger Möbius ladders. Note that each M_n can be drawn this way, centralizing two $(\frac{n}{2}+1)$-gons. Such a triangulation for M_{10} is given in Figure 5.11. Here, all lines have length 1, since all triangles in the triangulation are equilateral. Indeed, M_{10} is bipartite, so this result was expected.

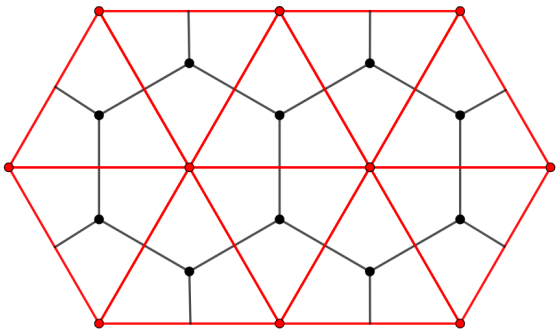


Figure 5.11: A 2-flow triangulation of M_{10}, where M_{10} is shown in black, and the triangulation in red.

For M_{12}, a similar triangulation reveals that

$$\phi_2(M_{12}) \leq 1 + \frac{1}{2\sin(\frac{2}{15}\pi)} = 2.2293$$

However, for $n \geq 14$, these triangulations yield upper bounds for $\phi_2(M_n)$ larger than $1+\sqrt{2}$, which are not useful.

5.3. r-flow triangulation of cubic bipartite graphs

In [8], an open problem is the question whether a "nice" flow triangulation exists for bipartite cubic graphs. Here, "nice" is defined in the following way:

Definition 6. *A flow triangulation is called **nice** if*

1. *the intersection of any two different triangles T_1 and T_2, if not empty, consists of either one vertex, or of two coinciding sides s_1 and s_2 of T_1 and T_2 respectively, such that s_1 and s_2 correspond to the same edge of G;*

2. *the set of all such edges induces a connected spanning subgraph of G.*

Note that the second statement is equivalent to the statement that the triangulation has to be connected; it cannot consist of multiple 'pieces'. Indeed, if the triangulation is not connected, the set of coinciding sides does not induce a connected subgraph. Furthermore, if the set of all such edges does not induce a spanning subgraph of G, then some vertex of G has no edges for which its two triangle sides coincide. In that case, triangle T_v is not connected to any other triangle, and the triangulation is not connected.

The goal of this section is to attempt to solve the open problem, and determine whether a nice flow triangulation exists for bipartite cubic graphs. There actually exists a simple flow triangulation for each cubic bipartite graph, but it is not necessarily nice. Of course, this follows directly from Theorem 10 and Theorem 18, but we want to explicitly define one. The idea is that we can draw an upward pointing equilateral triangle for every vertex of V_1, and a downward pointing equilateral triangle for every vertex of V_2, where $\{V_1, V_2\}$ is the vertex partition of the bipartite cubic graph.

Theorem 19. *Let G be a bipartite cubic graph. Then there exists a 2-flow triangulation of G.*

Proof. Let $\{V_1, V_2\}$ be the partition of the vertices of G. For each vertex $v_1 \in V_1$, we draw a line somewhere in the Euclidean plane that has length 1 and is parallel to the x-axis. Then, we draw two more lines such to form an equilateral triangle on the upper side of the first line. Since $|V_1|$ is finite, and the Euclidean plane is infinite, we can do this in such a way that no two triangles coincide. For each vertex $v_2 \in V_2$, we do the same, but now we draw the triangles on the lower side of the line that is parallel to the x-axis. An example of this, for the graph $K_{3,3}$, is given in Figure 5.12.

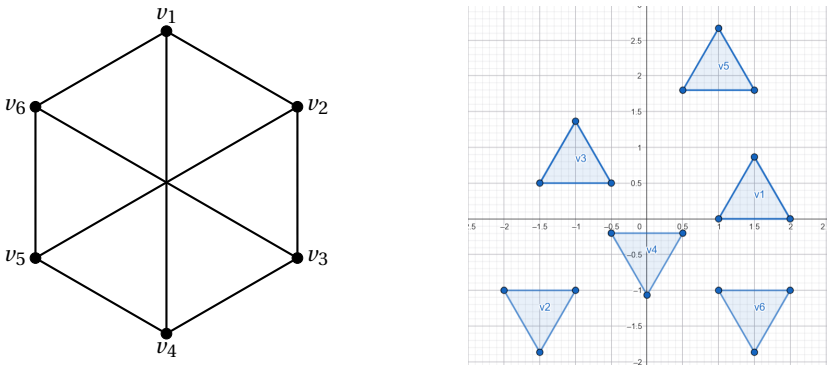


Figure 5.12: A 2-flow triangulation of $K_{3,3}$.

This is a flow triangulation. Indeed, the first two conditions of Definition 5 are trivially satisfied. That leaves us to prove that the sides of two different triangles that correspond to the same edge are attachable. By Lemma 4, bipartite cubic graphs are 3-edge-colourable. We colour the three edges of each triangle as in Figure 5.13.

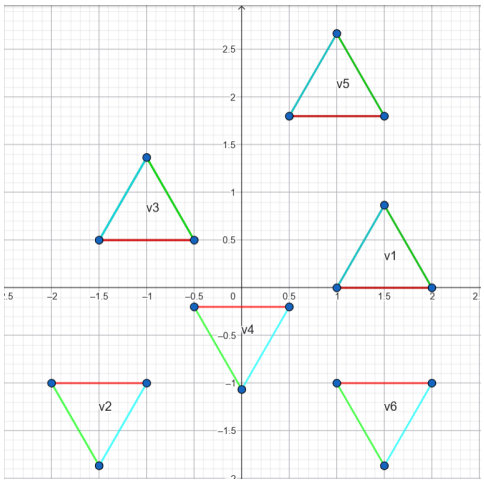


Figure 5.13: A colouring of the sides of the triangulation, corresponding to the 3-edge-colouring of G.

We use these three colours to make a 3-edge-colouring of G. For each vertex e incident to v, the triangle side of v corresponding to e will be the one with e's colour. It follows directly that each pair of sides of different triangles corresponding to the same edge is attachable. Thus, we have formulated a 2-flow triangulation of G. □

Unfortunately, this is not yet a "nice" triangulation. Since no two triangles have any intersecting points, there are no two coinciding sides s_1 and s_2 of different triangles T_1 and T_2 that correspond to the same edge of G, so the set of all such edges is empty. Evidently, an empty edge set does not induce a connected spanning subgraph of G. However, we can place the triangles in the Euclidean plane in such a way that the intersection of two triangles is not always empty. But can we do this in such a way that all coinciding sides induce a connected spanning subgraph?

If the cubic bipartite graph G contains a Hamilton cycle, such a construction always exists. In such a case, all triangles can be arranged in a linear sequence. An example of this is illustrated in Figure 5.14. In this figure, $v_1, v_2, \cdots, v_8$ is a Hamiltoncycle. It indeed holds that for each $i \in \{2, \cdots, 8\}$, triangle v_i is attachable (and in this case, attached) to triangle v_{i-1} in the triangulation. Furthermore, the last triangle v_8 is also attachable to v_1; this works for all cubic bipartite graphs G with a Hamilton cycle, since these graphs must have an even number of vertices. Furthermore, the remaining edges are represented by the horizontal lines in Figure 5.14. They are all attachable to all horizontal sides of triangles representing a vertex in the other set of the partition. Therefore, this is a nice flow triangulation for a cubic bipartite graph containing a Hamilton cycle.

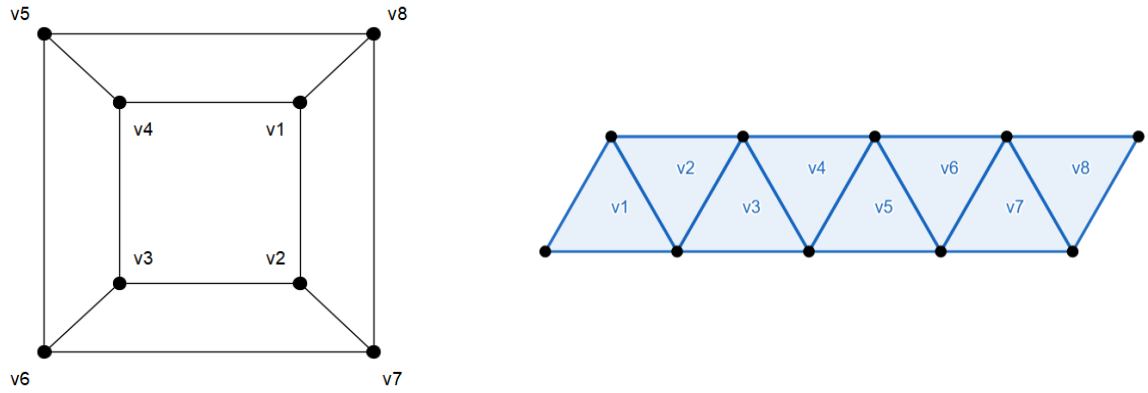


Figure 5.14: The cube graph (left) is an example of a cubic biparte graph with a Hamilton cycle. Therefore, we can draw a nice 2-flow triangulation (right).

The question remains whether a nice flow triangulation also exists for cubic bipartite graphs without a Hamilton cycle. Most simple examples of cubic bipartite graphs contain Hamilton cycles. However, the graph in Figure 5.15, does not. This graph consists of two middle vertices, numbered 19 and 20, and three parts that are only connected to each other through 19 and 20. This means that a Hamilton cycle has to go through the set of middle vertices three times, but that set only contains the two vertices 19 and 20. That gives a contradiction, so we can conclude this graph does not contain a Hamilton cycle. It is, in fact, the smallest cubic bipartite graph with no Hamilton cycle.

This graph does have a nice 2-flow triangulation, which is shown in Figure 5.16. This shows that there are also non-Hamiltonian cubic bipartite graphs that have a nice flow triangulation. Another example of a non-Hamiltonian cubic bipartite graph that has a nice flow triangulation is shown in Figure 5.17. This, however, does not prove that all non-Hamiltonian cubic bipartite graphs have a nice flow triangulation. Therefore, the open problem is not yet solved. The remaining challenge is to solve this for all non-Hamiltonian cubic bipartite graphs.

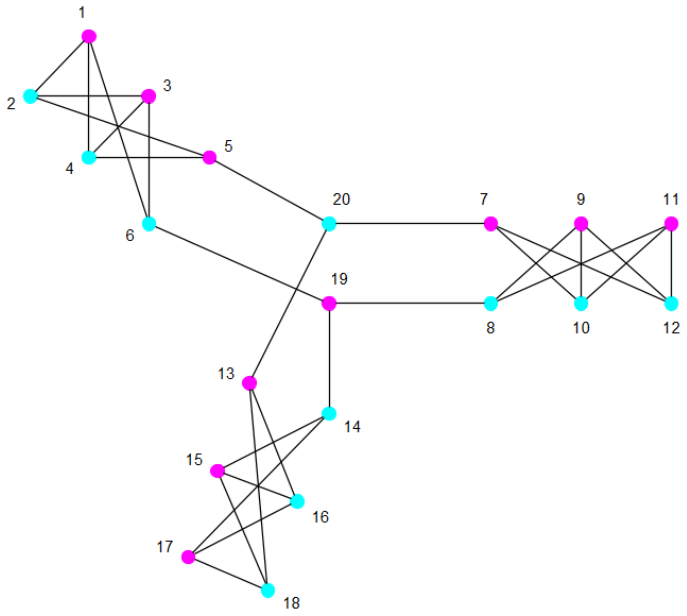


Figure 5.15: An example of a cubic bipartite graph with no Hamilton cycle.

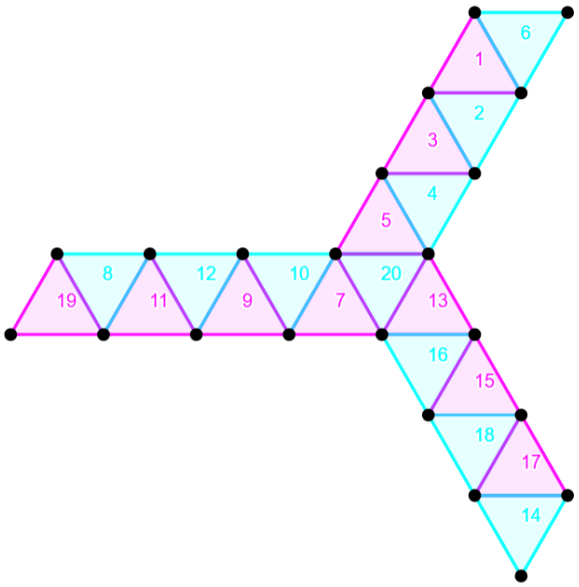


Figure 5.16: A nice 2-flow triangulation for the cubic bipartite graph with no Hamilton cycle from Figure 5.15.

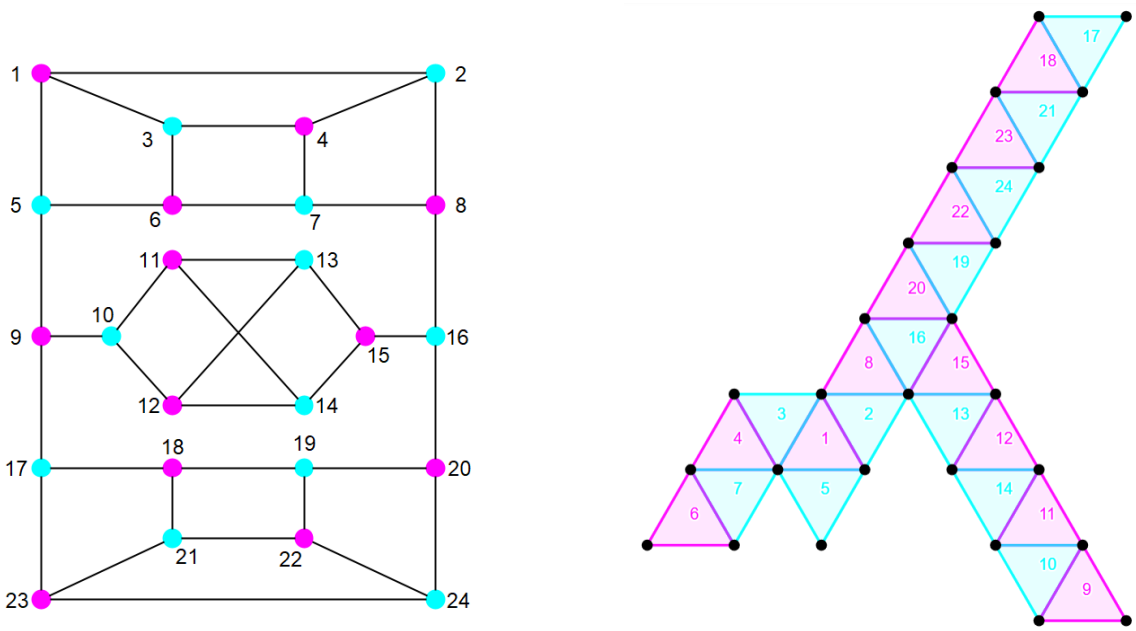


Figure 5.17: A cubic bipartite graph with no Hamilton cycle that has 24 vertices (left) with a nice flow triangulation (right).

6

Approximating two-dimensional flow numbers with optimization models

The goal of this section is to use optimization models to approximate two-dimensional flow numbers of certain graphs. In Section 6.1 and Section 6.2, the goal is to find a certain two-dimensional nowhere-zero r-flow on the Wagner graph and Petersen graph respectively, since such an r is an upper bound to the two-dimensional flow number of the graph. This is done using linear programming. In Section 6.3, a general flow optimization model is created to find even better flows on these and other graphs.

6.1. Finding a two-dimensional flow on the Wagner graph

In Section 5.2, we determined that $2.1755 \leq \phi_2(M_8) \leq 1.2856$ for the Wagner graph M_8. In this section, the goal is to improve this upper bound by explicitly finding a two-dimensional nowhere-zero r-flow on the Wagner graph. We define the flow in Figure 6.1, where a, b, c and d are elements of $\mathbb{R}^2$.

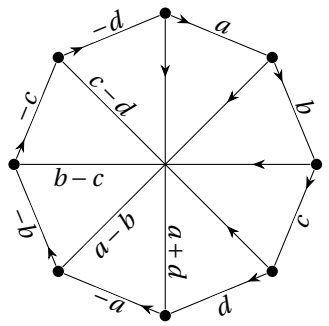


Figure 6.1: A flow on the Wagner graph, where a, b, c and d are from $\mathbb{R}^2$.

It is not clear whether this is the optimal way to define a flow on the Wagner graph. However, if we find a certain r-flow, it is certain that r is an upper bound of the two-dimensional flow number of the Wagner graph, even if it is not the best possible upper bound.

We now assume that a, b, c and d all have absolute value 1, which is equivalent to them being on the unit circle S^1. Note that this assumption is not without loss of generality; it does, however, make the modelling easier. The goal is to place a, b, c and d on the unit circle in such a way that the maximum of the absolute values of $a-b$, $b-c$, $c-d$ and $-a-d$ is minimized. That is equivalent to placing a, b, c and d on the unit circle in such a way that the maximum of the distances $\|a-b\|$, $\|b-c\|$, $\|c-d\|$ and $\|-a-d\|$ is minimized. Note that each of these distances has to be at least 1, by of the definition of d-dimensional nowhere-zero flows. An example of a feasible placement of b, c and d is drawn below in Figure 6.2.

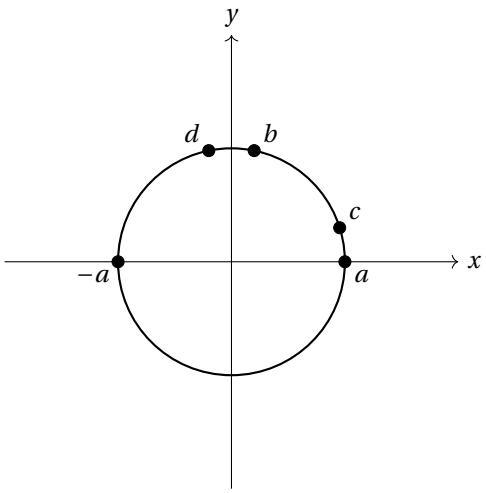


Figure 6.2: An example of a feasible placement of b, c and d.

Note that for each solution, the circle can be turned by multiplying every value by the same complex constant in such a way that a is at $(1,0)$ in the xy-plane. Therefore, placing a at $(1,0)$ is no restriction. This makes a a constant instead of a variable, making this a problem in the variables b, c and d. To solve this optimization problem, linear programming will be used. Unfortunately, the problem is not linear yet, as it contains absolute values and the minimazation of a maximum. Also, right now, b, c and d are elements of $\mathbb{R}^2$, not $\mathbb{R}$. To fix this, the variables in the model will be the angles:

$$\begin{aligned}\beta &= \angle(a,b)\\ \gamma &= \angle(a,c)\\ \delta &= \angle(a,d)\end{aligned}$$

We also assume that b, c and d are in the upper half of the unit circle. Therefore, b, c and d are real numbers between 0 and 180. This way we get the following Mixed Integer Problem (MIP). First, the variables are defined and explained in the variable definition, and then the model is presented.

Variable definition

$$\begin{aligned}&\beta = \angle(a,b) \in [0,180]\\ &\gamma = \angle(a,c) \in [0,180]\\ &\delta = \angle(a,d) \in [0,180]\\ &z = \max\{\beta, |\gamma-\beta|, |\delta-\gamma|, 180-\delta\}\\ &y_1 = \begin{cases}1 & \text{if } \gamma \geq \beta\\ 0 & \text{if } \beta > \gamma\end{cases}\\ &y_2 = \begin{cases}1 & \text{if } \delta \geq \gamma\\ 0 & \text{if } \gamma > \delta\end{cases}\end{aligned}$$

Model

$$\begin{aligned}\min\ & z \quad \text{s.t.}\\ & \beta \geq 60\\ & \beta - \gamma - 60 \geq -400 y_1\\ & \gamma - \beta - 60 \geq -400(1-y_1)\\ & \gamma - \delta - 60 \geq -400 y_2\\ & \delta - \gamma - 60 \geq -400(1-y_2)\\ & \delta \leq 120\\ & \beta \leq z\\ & \gamma - \beta \leq z\\ & \beta - \gamma \leq z\\ & \delta - \gamma \leq z\\ & \gamma - \delta \leq z\\ & 180 - \delta \leq z\end{aligned}$$

The optimal solution of this LP is found using Python. The code can be found in the appendix, in A.1. The following solution was found to be optimal, which is drawn on the unit circle in Figure 6.3.

Optimal solution

$\beta = 80^\circ$
$\gamma = 20^\circ$
$\delta = 100^\circ$

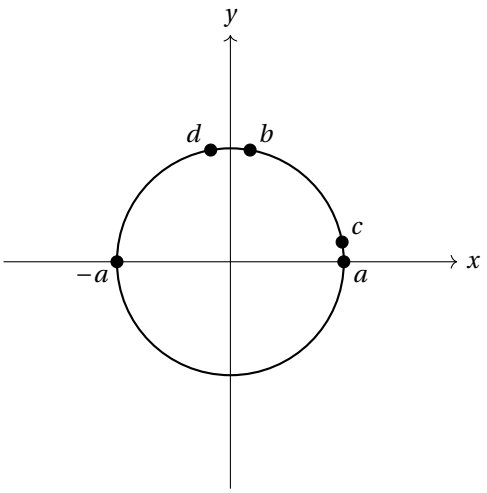


Figure 6.3: The optimal solution, drawn on the unit circle.

The maximum of the distances $\|a-b\|$, $\|b-c\|$, $\|c-d\|$ and $\|-a-d\|$ is now equal to

$$\|a-b\| = \|b-c\| = \|c-d\| = \|-a-d\| = \sqrt{(1-\cos(80^\circ)^2 + \sin(80^\circ)^2} = 2\sin(40^\circ) \approx 1.28557522$$

This equals the upper bound found in Section 5.2 using flow triangulations.

6.2. The Petersen graph

The goal of this section is to extend the method of Section 6.1 to the Petersen graph. In [8], it is proved that the two-dimensional flow number of the Petersen graph is at most $1+\sqrt{7/3}$. Furthermore, it is conjectured that this is its two-dimensional flow number, and not just an upper bound. We define the the flow on the Petersen graph that is presented in Figure 6.4.

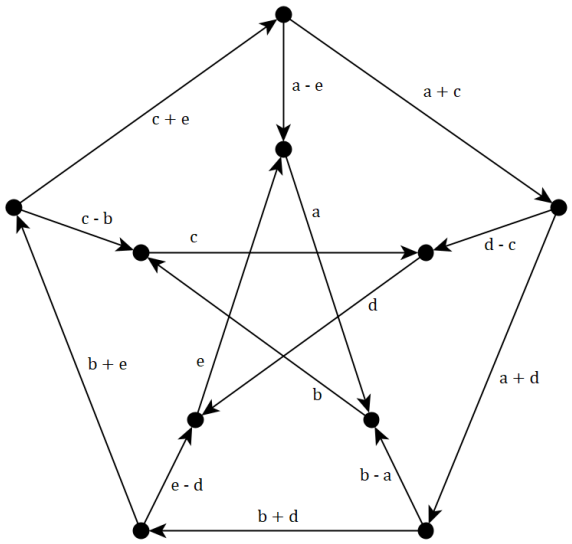


Figure 6.4: A flow on the Petersen graph, where a, b, c, d and e are from $\mathbb{R}^2$.

Again, we assume that a, b, c, d and e are on the unit circle, which is not without loss of generality. Also, we place a at the point $(1,0)$ again; that is without loss of generality. Now, we want to minimize the maximal flow value. In other words, we want to minimize

$$\max\{\|a-b\|, \|b-c\|, \|c-d\|, \|d-e\|, \|e-a\|, \|-c-a\|, \|-d-a\|, \|-d-b\|, \|-e-b\|, \|-e-c\|\}.$$

The main constraint is that each of these distances has to be greater than or equal to 1. We formulate this problem as an MIP too. Again, to eliminate norms, we will use angles, defined in degrees:

$$\begin{aligned} \beta &= \angle(a,b) & \beta' &= \angle(a,-b) \\ \gamma &= \angle(a,c) & \gamma' &= \angle(a,-c) \\ \delta &= \angle(a,d) & \delta' &= \angle(a,-d) \\ \epsilon &= \angle(a,e) & \epsilon' &= \angle(a,-e) \end{aligned}$$

In this model, we need to look at the whole circle, instead of only the upper half, since this time not only $-a$ appears, but also $-b$, $-c$, $-d$ and $-e$. Therefore, all angles are between $-180°$ and $180°$. Unfortunately, this complicates the linearization of our problem significantly. Take γ as an example. As we want γ' to be the angle between a and $-c$, γ' depends on γ in the following way:

$$\gamma' := \begin{cases} \gamma - 180 & \text{if } \gamma \geq 0 \\ \gamma + 180 & \text{if } \gamma < 0 \end{cases}$$

This is not yet linear, so it cannot be used in the MIP. To fix this, we split the MIP into sixteen cases. If we assume beforehand in which half of the circle b, c, d and e have to be placed, we can define b', c', d' and e' in a way that keeps the problem linear. We know that each b, c, d, e either has to be in the upper half of the circle or in the lower half. Therefore, there are sixteen possibilities. The idea is to find the optimal solution for each of these sixteen cases, and then choose the best (minimal) solution out of these sixteen. To model this in Python, a variable i is introduced. The code uses a for-loop to find the result for each $i \in \{0, \cdots, 15\}$. To define β', γ', δ' and ϵ' for each i, the following part of code is used.

```
if i < 8:
    model += beta >= 0
    beta_ = beta - 180
else:
    model += beta <= 0
    beta_ = beta + 180

if i // 4 == 0 or i // 4 == 2:
    model += gamma >= 0
    gamma_ = gamma - 180
else:
    model += gamma <= 0
    gamma_ = gamma + 180

if (i // 2) % 2 == 0:
    model += delta >= 0
    delta_ = delta - 180
else:
    model += delta <= 0
    delta_ = delta + 180

if i % 2 == 0:
    model += epsilon >= 0
    epsilon_ = epsilon - 180
else:
    model += epsilon <= 0
    epsilon_ = epsilon + 180
```

Then, the contraints can be modelled. The challenge here is to ensure that the right distance is used. For example, if $\beta = 120$ and $\gamma = -120$, their distance is 60 degrees, and not 240. To define this mathematically, for angles $i, j \in \{\beta, \gamma, \delta, \epsilon, \beta', \gamma', \delta', \epsilon'\}$, it should hold that

$$\begin{aligned} d(i,j) &= \min\{|i-j|, 360 - |i,j|\} \\ &= \min\{\max\{i-j, j-i\}, 360 - \max\{i-j, j-i\}\} \end{aligned}$$

To model this in Python, we first introduce a variable `dis_i_j`, which we want to be equal to $|i-j|$. Then, we introduce `dis_i_j_real` to model $\min\{|i-j|, 360-|i,j|\}$. Finally, we add the constraints that `dis_i_j_real` must be ≥ 60 and $\leq z$ for all i and j, to make z our maximum distance, and all distances at least 1. This is done in the following way for β and γ.

```
M = 400  # big-M constant
# beta - gamma
model += dis_beta_gamma >= beta - gamma
model += dis_beta_gamma >= gamma - beta
model += gamma - beta - dis_beta_gamma >= -M * w[1]
model += beta - gamma - dis_beta_gamma >= -M * (1 - w[1])
model += dis_beta_gamma_real <= dis_beta_gamma
model += dis_beta_gamma_real <= 360 - dis_beta_gamma
model += dis_beta_gamma_real - dis_beta_gamma >= -M * v[1]
model += dis_beta_gamma_real - (360 - dis_beta_gamma) >= -M * (1 - v[1])
model += dis_beta_gamma_real >= 60
model += dis_beta_gamma_real <= z
```

Of course, this is repeated for all other distances. However, we still have to make sure that $|\beta|$, $|\epsilon|$, $|\gamma'|$ and $|\delta'|$ are greater than or equal to 60 and less than or equal to z. Note that $|\gamma'| \geq 60$ is equivalent to $|\gamma| \leq 120$. This can be obtained by simply changing γ's bounds to $\gamma \in [-120, 120]$ instead of $(-180, 180]$. The same is done for δ. The other constraints are modelled in the following way:

```
# Other
model += epsilon - 60 >= -M * y[1]
model += -epsilon - 60 >= -M * (1 - y[1])
model += beta - 60 >= -M * y[2]
model += -beta - 60 >= -M * (1 - y[2])

model += z >= beta
model += z >= -beta
model += z >= epsilon
model += z >= -epsilon
model += z >= gamma_
model += z >= -gamma_
model += z >= delta_
model += z >= -delta_
```

Now, the code can be completed. The full code can be found in the appendix, in Section A.2. It gives the result $z = 150$, which means we have found a flow with maximal flow value $2\sin(75°) \approx 1.9319$. Therefore, we have obtained the upper bound 2.9319 for the two-dimensional flow number of the Petersen graph. This is way higher than the already known upper bound $1 + \sqrt{7/3} \approx 2.5275$, so it appears that the flow defined in Figure 6.4 was no optimal choice.

6.3. A general flow optimization model

In the previous sections, linear programming was used to determine a certain two-dimensional flow on the Wagner graph (Section 6.1) and the Petersen graph (Section 6.2). In this section, the goal is to construct a more general model where all flow values are variables, rather than only a few.

The problem of finding an optimal two-dimensional flow on a certain graph can be reformulated as a problem in which we have to place certain points in the two-dimensional plane. A two-dimensional flow value is a vector consisting of two components, which can also be represented as a point in the xy-plane. Furthermore, a two-dimensional r-flow is a placement of points in the plane such that:

1. all points lie outside the circle around O with radius 1;
2. all points lie inside the circle around O with radius r;

3. all points satisfy the graph constraints.

The graph constraints are to ensure Kirchhoff's law holds at every vertex. If a vertex v, for example, has ingoing edges e_1 and e_2 and outgoing edge e_3, and the corresponding points in the xy-plane are z_1, z_2 and z_3 respectively, then the 'graph constraint' $z_1 + z_2 - z_3 = 0$ must be added to ensure that v satisfies Kirchhoff's law. The first two constraints are visualized in Figure 6.5.

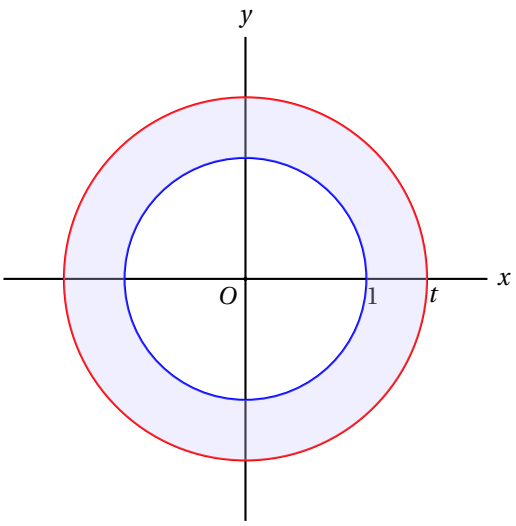


Figure 6.5: A two-dimensional r-flow is equivalent to a certain placement of points in this xy-plane, such that all points lie between the blue circle with radius 1 and red circle with radius t.

We obtain the following model:

$$\begin{aligned} &\min t \text{ s.t. } x_i \text{ and } y_i \text{ satisfy graph constraints, and} \\ &x_i^2 + y_i^2 \geq 1 \\ &x_i^2 + y_i^2 \leq t^2 \end{aligned}$$

Unfortunately, this model is quadratic and not linear. Furthermore, the solution set is non-convex. A solution set to an optimization problem is called *convex* if for every two feasible solutions x and y, all points on the line between x and y are feasible solutions as well. In Figure 6.5, one can clearly see why this model has a non-convex solution set; $(0,1)$ and $(0,-1)$ are feasible solutions, but $(0,0)$ is not, even though it is on the line from $(0,1)$ to $(0,-1)$. Furthermore, an optimization problem is called *convex* if both the solution set and the objective function are convex. Note that our objective function is convex, but since the solution set is non-convex, so is the problem. Many solvers for optimization problems rely on convexity.

To still find solutions, the software Gurobi is used. Since the running time is very long, it is not possible to find the optimal solution. However, each solution found is an upper bound to the two-dimensional flow number of the graph. Therefore, we run the program for a fixed period of time. The obtained solution may not be the optimal flow on graph G, but it does give an upper bound to $\phi_2(G)$. These can be compared to lower bounds, possibly found by Theorem 11.

The Python code for this model is included in Appendix A.3. In the following subsections, certain graphs are presented, on which this model is used. In the first two subsections, the model will be used on the Wagner graph and the Petersen graph, to verify and possibly improve earlier results on these. The next three are about snarks, since very little is known about these graphs.

6.3.1. The Wagner graph

Recall from Section 5.2 and Section 6.1 that $2.1756 \approx 1 + 2\sin\left(\frac{\pi}{5}\right) \leq \phi_2(M_8) \leq 1 + 2\sin\left(\frac{2\pi}{9}\right) \approx 2.2856$. Running the code for 10 minutes gives a two-dimensional nowhere zero 2.2856-flow, a result that was already found after one minute. Furthermore, when we impose the constraint $t \leq 2.285$ on the model, the code returns

that the model is infeasible, already after 40 seconds. This implies that $2.285 \leq \phi_2(M_8) \leq 2.2856$. The results presented in this section, together with those from Section 5.2 and Section 6.1, provide sufficient justification for the following conjecture.

Conjecture 3. *Let M_8 be the Wagner graph. Then $\phi_2(M_8) = 1 + 2\sin\left(\frac{2\pi}{9}\right)$.*

6.3.2. The Petersen graph

The goal of this subsection is to use our code to support or disprove the conjecture of [8] that states that $\phi_2(P) = 1 + \sqrt{7/3}$ for the Petersen graph P. After running the code for 10 minutes, the best flow found was a two-dimensional $1 + \sqrt{7/3}$-flow; this result was found after 4 minutes. Furthermore, adding a constraint $t \leq 1.5$ makes the model infeasible, a result that was found after less than 3 minutes. We have computationally shown that $2.5 \leq \phi_2(P) \leq 2.5276 \approx 1 + \sqrt{7/3}$. Note that this also proves that the Petersen graph has two-dimensional flow number larger than $1 + \sqrt{2}$. The results of this section support the conjecture that $\phi_2(P) = 1 + \sqrt{7/3}$.

6.3.3. Tietze's graph

Tietze's graph, denoted by T in this section, is the smallest snark after the Petersen graph. It can be constructed from the Petersen graph by replacing one of the vertices by a triangle. It is shown in Figure 6.6. A two-dimensional nowhere-zero r-flow on Tietze's graph can be used to construct an r-flow on the Petersen graph; simply contract the vertices of the triangle. Therefore, if we denote the Petersen graph by P, we obtain $\phi_2(T) \geq \phi_2(P)$. The conjecture that $\phi_2(P) = 1+\sqrt{7/3}$ would therefore imply that $\phi_2(T) \geq 1+\sqrt{7/3}$. Running the code from A.3 for 90 minutes gives a two-dimensional flow with maximal norm $r = 1.52752 = \sqrt{7/3}$; this result was already found after 15 minutes. Thus, we have computationally proved that $\phi_2(T) \leq 1 + \sqrt{7/3}$. Therefore, the following conjecture follows directly from the conjecture that $\phi_2(P) = 1 + \sqrt{7/3}$, and is supported by the fact that the computation did not improve after 15 minutes.

Conjecture 4. $\phi_2(T) = \phi_2(P) = 1 + \sqrt{7/3}$.

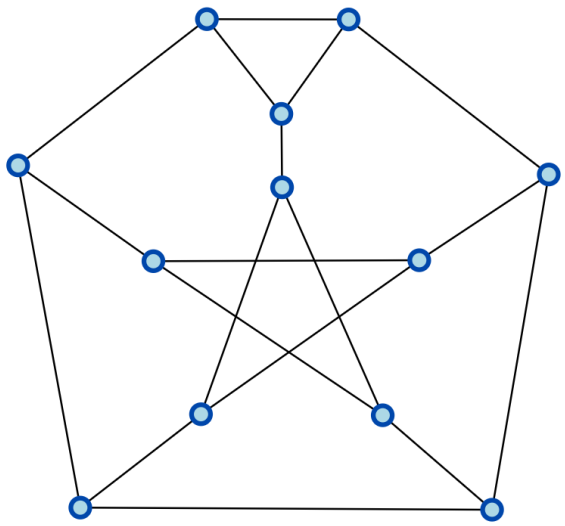


Figure 6.6: Tietze's graph, from [1].

6.3.4. Another snark

Another relatively small snark is shown in Figure 6.7. It has fourteen vertices. After running the code for 90 minutes on this graph, the best flow found has $t = 1.52753$, so for this snark S it holds $\phi_2(S) \leq 2.52753$. This upper bound is greater than $1 + \sqrt{2}$, so it is not clear whether this snark has two-dimensional flow number smaller than $1 + \sqrt{2}$. The result was already found after 24 minutes.

6.3.5. The flower snark J_5

In Figure 6.8, the flower snark J_5 is presented. It was discovered by Isaacs in 1975 and has 20 vertices [4]. In [8], an r-flow triangulation of this snark is presented to show that $\phi_2(J_5) \leq 2.387893647$. While 3-edge-colourable cubic graphs G have the property $\phi_2(G) \leq 1+\sqrt{2}$ by Theorem 13, snarks do not have this property, and the Petersen graph likely has $\phi_2(G) = 1+\sqrt{7/3} > 1+\sqrt{2}$. Therefore, this upper bound of $\phi_2(G)$ is interesting, as is shows that not all snarks have two-dimensional flow number greater than $1+\sqrt{2}$.

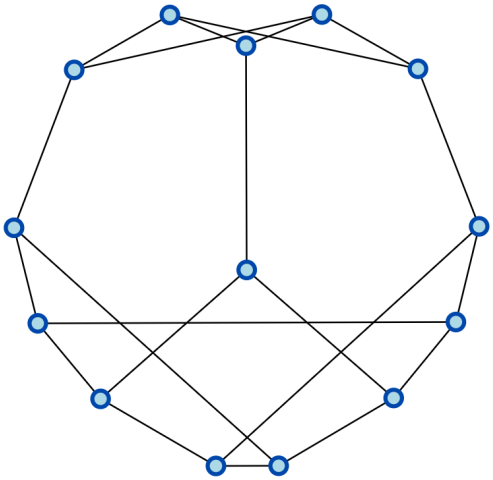


Figure 6.7: Another small snark, from [1].

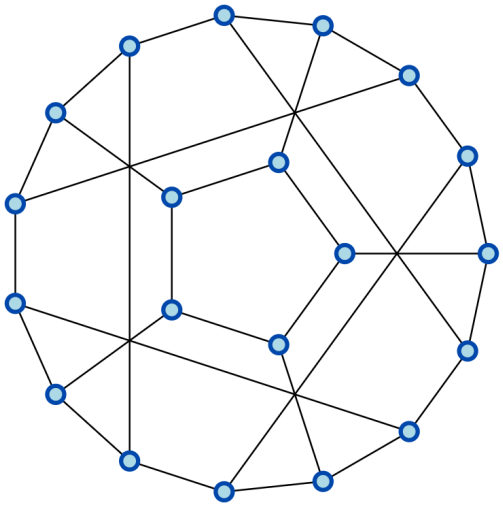


Figure 6.8: The flower snark J_5, from [1].

This code was run for 20 hours (1200 minutes), as the result kept improving. The resulting upper bound is $\phi_2(J_5) \leq 2.41748$, which is a bit larger than $\sqrt{2}$. Thus, the model did not improve the upper bound $\phi_2(J_5) \leq 2.387893647$ from [8]. It appears that the code performs less effectively for larger graphs.

6.3.6. Bidiakis cube

In Figure 6.9, Bidiakis cube is shown. It is a 3-edge-colourable cubic graph, containing induced cycles of lengths 4 and 5. By Theorem 11, we know that $\phi_2(B) \geq 2.1755705$; we use B to denote Bidiakis cube in this subsection. Furthermore, by Theorem 13, $\phi_2(B) \leq 1+\sqrt{2}$.

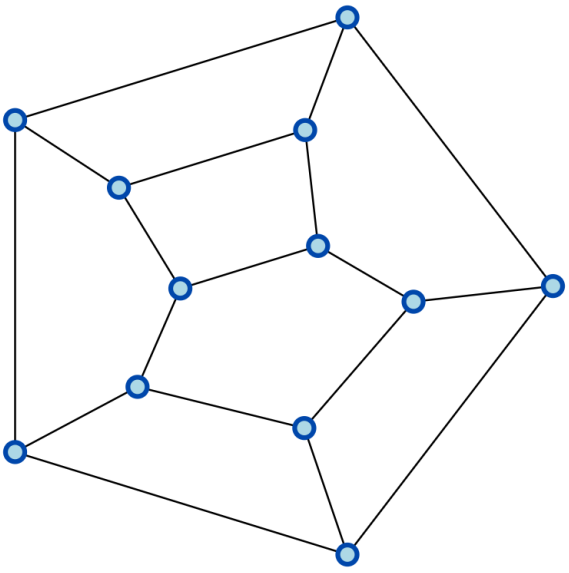


Figure 6.9: Bidiakis cube, from [1].

Running the code for 90 minutes gave a nowhere-zero 2.33629-flow on Bidiakis cube, thus $\phi_2(B) \leq 2.33629$. This improves the upper bound obtained from Theorem 13.

In Table 6.1, all results of these subsections are summarized. Since the exact two-dimensional flow number is not yet determined for any of the graphs of these subsections, the table contains upper bounds, lower bounds and conjectured two-dimensional flow numbers.

Graph (G)	**Figure**	**Lower bound to** $\phi_2(G)$	**Upper bound to** $\phi_2(G)$	**Conjectured** $\phi_2(G)$
Wagner graph M_8	Figure 5.3	2.285	$1+2\sin\left(\frac{2\pi}{9}\right)$	$1+2\sin\left(\frac{2\pi}{9}\right)$
Petersen graph P	Figure 2.5	2.5	$2+\sqrt{7/3}$	$1+\sqrt{7/3}$
Tietze's graph T	Figure 6.6	$\phi_2(P)$	$1+\sqrt{7/3}$	$1+\sqrt{7/3}$
Another snark S	Figure 6.7	-	2.52753	-
The flower snark J_5	Figure 6.8	-	2.38789	-
Bidiakis cube B	Figure 6.9	$1+2\sin\left(\frac{\pi}{5}\right)$	2.33629	-

Table 6.1: An overview of the results for the different graphs investigated in the subsections. The upper bounds to $\phi_2(P)$ and $\phi_2(J_5)$ are from [8], as well as the conjecture for $\phi_2(P)$.

6.4. Another approach: approximating with polygons

It becomes clear that the code in Appendix A.3 used in Section 6.3 has very long running time, especially for larger graph; the optimal two-dimensional nowhere-zero flow on the flower snark J_5 was not yet found after 20 hours. This is (partially) due to the nonlinearity of the problem; since it is a quadratic optimization problem, solving it requires significantly more computational time. One way to avoid this issue is to replace the circles with regular n-gons. The circle of radius 1 could be approximated by its circumscribed polygon, and the circle of radius t by its inscribed polygon. An example of this is presented in Figure 6.10.

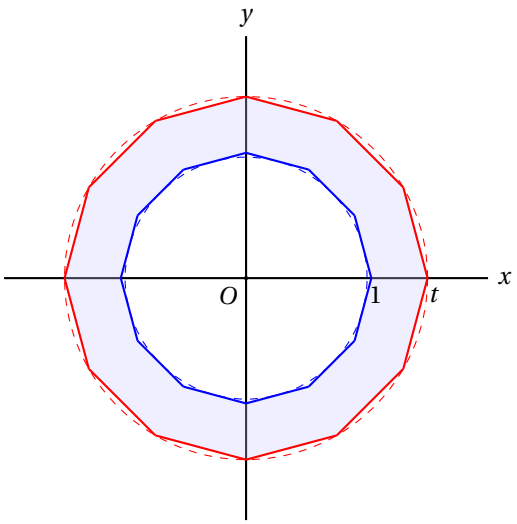


Figure 6.10: A linear approximation of the original problem, using dodecagons.

This way, each point (x_i, y_i) still lies outside the circle of radius 1 and inside the circle of radius t. Furthermore, all constraints become linear. Thus, each solution of this problem is also a solution of the original problem. However, it is not necessarily the optimal solution of the original problem; it only approximates it. Yet, this linearization could lower the running time, and for large n still return accurate approximations. Therefore, this merits further research.

7

Conclusions

The goal of this thesis was to further research two-dimensional flow numbers for certain graphs. In Chapter 2, all necessary concepts about this subject are introduced or defined. In Chapter 3, relevant literature was consulted to give an overview of the existing theorems and conjectures on multi-dimensional flow numbers, and some proofs were rewritten to clearify them. These conjectures and theorems could be used in further chapters, which aimed to extend the existing theory.

In Chapter 4, the goal was to determine the two-dimensional flow number for all complete multipartite graphs. This goal was realized; the result is that the two-dimensional flow number for all complete multipartite graphs is 2, except for K^4; for this graph, $\phi_2(K^4) = 1+\sqrt{2}$. This is an unexpected and notable result. To obtain this, first, the two-dimensional flow number was determined for all complete graphs using theorems from Chapter 3. Then, an induction proof was used to show the result for all complete bipartite graph. Lastly, this was used to show some propositions, which together covered all complete multipartite graphs.

In Section 5.2, a $2\sin\left(\frac{2}{9}\pi\right)$-flow triangulation was found for the Wagner graph M_8. It was concluded that $2\sin\left(\frac{1}{5}\pi\right) \leq \phi_2(M_8) \leq 2\sin\left(\frac{2}{9}\pi\right)$. Extending this flow triangulation to larger Möbius ladders did not yield any new results. Furthermore, in Section 5.3, the goal was to find a 'nice' flow triangulation for all bipartite cubic graphs. The desired outcome was not attained, but we did find nice flow triangulations for Hamiltonian cubic bipartite graphs, and for two non-Hamiltonian cubic bipartite graphs.

In Chapter 6, the goal was to computationally approximate two-dimensional flow numbers of certain graphs. First, optimal values for specific flows on the Wagner and Petersen graph were found using linear programming. The obtained flow value for the Wagner graph was equal to the one found in Section 5.2 through flow triangulations, while the obtained flow value for the Petersen graph was larger than the already known upper bound of Theorem 15.

Second, we aimed to formulate a general flow optimization model, and use it to approximate the two-dimensional flow number of certain graphs, namely the Wagner graph, Petersen graph, Tietze's graph, another snark, the flower snark J_5 and Bidiakis cube. The code of Appendix A.3 serves this purpose, but has very long running time. Nevertheless, multiple results were found; they were presented in Table 6.1. The computational results of Subsection 6.3.2 further support the conjectured value $\phi_2(P) = 1+\sqrt{7/3}$ of the Petersen graph. Furthermore, the results of Subsection 6.3.1 and Subsection 6.3.3 provided sufficient justification for Conjecture 3 and Conjecture 4, which respectively state that $\phi_2(M_8) = 1+2\sin\left(\frac{2\pi}{9}\right)$ and $\phi_2(T) = 1+\sqrt{7/3}$ for the Wagner graph M_8 and Tietze's graph T. The upper bound of the two-dimensional flow number of the flower snark J_5 from [8] was not improved by the computational results.

A suggestion for further research is presented in Subsection 6.4; this method could lower the running time of the code of Appendix A.3, hence find better results for larger graphs such as the flower snark J_5.

Bibliography

[1] House of graphs – a searchable database of graphs, 2025. URL https://houseofgraphs.org/. Extensive database used for graph information and images.

[2] Jean-Claude Bermond, Bill Jackson, and François Jaeger. Shortest coverings of graphs with cycles. *Journal of Combinatorial Theory, Series B*, 1983.

[3] Reinhard Diestel. *Graph Theory*. Springer, Berlin, Heidelberg, 6th edition, 2017. ISBN 978-3-662-53622-6.

[4] Rufus Isaacs. Infinite families of nontrivial trivalent graphs which are not tait colorable. *The American Mathematical Monthly*, pages 221–239, 1975. URL https://www.jstor.org/stable/2319844.

[5] Chun-Hung Liu. Tutte's flow conjectures. Lecture notes, Texas A&M University, 2022. URL https://people.tamu.edu/~chliu/teaching/spring22math689/2022_02_22.pdf. Lecture dated February 22, 2022.

[6] L. M. Lovász, Carsten Thomassen, Yezhou Wu, and C.-Q. Zhang. Nowhere-zero 3-flows and modulo k-orientations. *Journal of Combinatorial Theory, Series B*, 2013. URL https://math.wvu.edu/~cqzhang/Publication-files/my-paper/JCTB-2013-3-flow-LTWZ.pdf.

[7] Davide Mattiolo, Giuseppe Mazzuoccolo, Jozef Rajník, and Gloria Tabarelli. A lower bound for the complex flow number of a graph: a geometric approach. 2023. URL https://arxiv.org/abs/2303.10281.

[8] Davide Mattiolo, Giuseppe Mazzuoccolo, Jozef Rajník, and Gloria Tabarelli. On d-dimensional nowhere-zero r-flows on a graph. *European Journal of Mathematics*, 2023. URL https://link.springer.com/article/10.1007/s40879-023-00694-1.

[9] NRICH. The four colour theorem, 2011. URL https://nrich.maths.org/articles/four-colour-theorem. Accessed on October 31, 2025.

[10] P. D. Seymour. Nowhere-zero 6-flows. *Journal of Combinatorial Theory, Series B*, 1981. URL https://www.sciencedirect.com/science/article/pii/0095895681900587.

[11] Carsten Thomassen. Group flow, complex flow, unit vector flow, and the $(2+\epsilon)$-flow conjecture. *Journal of Combinatorial Theory, Series B*. URL https://backend.orbit.dtu.dk/ws/portalfiles/portal/128128802/group_flow.postscript.pdf.

[12] W. T. Tutte. A contribution to the theory of chromatic polynomials. *Canadian Journal of Mathematics*, 1954. URL https://www.cambridge.org/core/journals/canadian-journal-of-mathematics/article/contribution-to-the-theory-of-chromatic-polynomials/E1EE1F053B494D08746F0EE23F736CC2.

[13] Cun-Quan Zhang. *Integer Flows and Cycle Covers of Graphs*. Pure and Applied Mathematics. Marcel Dekker, Inc., New York, 1997. ISBN 0-8247-9790-6.

A

Appendix

A.1. Code for the MIP of the Wagner graph

The following code is used in Section 6.1 to find the optimal values of a certain two-dimensional nowhere-zero flow on the Wagner graph.

```
from pulp import *

# ---------------------------------------------------
# 1. Problem
# ---------------------------------------------------
model = LpProblem("Wagnergraph", LpMinimize)

# ---------------------------------------------------
# 2. Variables
# ---------------------------------------------------
b = LpVariable("b", lowBound=60, upBound=180, cat="Continuous") #angle between a and b
c = LpVariable("c", lowBound=0, upBound=180, cat="Continuous") #angle between a and c
d = LpVariable("d", lowBound=0, upBound=120, cat="Continuous") #angle between a and d
y1 = LpVariable("y1", cat="Binary")
y2 = LpVariable("y2", cat="Binary")
z = LpVariable("z", lowBound=0, cat="Continuous")

# ---------------------------------------------------
# 3. Constraints
# ---------------------------------------------------
model += b-c-60>=-400*y1
model += c-b-60>=-400*(1-y1)
model += d-c-60>=-400*y2
model += c-d-60>=-400*(1-y2)
model += b<=z
model += b-c<=z
model += c-b<=z
model += d-c<=z
model += c-d<=z
model += 180-d<=z

# ---------------------------------------------------
# 4.Objective function
# ---------------------------------------------------
model += z, "Objective"
```

```
# ------------------------------------------------
# 5. Solve
# ------------------------------------------------
model.solve(PULP_CBC_CMD(msg=True))

# ------------------------------------------------
# 6. Results
# ------------------------------------------------
print("Status:", LpStatus[model.status])
print("Optimal value:", value(model.objective))
print("b =", value(b))
print("c =", value(c))
print("d =", value(d))
```

A.2. Code for the MIP of the Petersen graph

The following code is explained and used in Section 6.2 to find the optimal values of a certain two-dimensional nowhere-zero flow on the Petersen graph.

```
from pulp import *
import pandas as pd

results = []

for i in range(16):
    model = LpProblem(Petersengraph_i{i}", LpMinimize)

    # ------------------------------------------------
    # 2. Variables
    # ------------------------------------------------
    beta = LpVariable("beta", lowBound=-180, upBound=180)
    gamma = LpVariable("gamma", lowBound=-120, upBound=120)
    delta = LpVariable("delta", lowBound=-120, upBound=120)
    epsilon = LpVariable("epsilon", lowBound=-180, upBound=180)

    # Distance variables (dis_...)
    dis_beta_gamma = LpVariable("dis_beta_gamma",
    lowBound=0, upBound=360)
    dis_gamma_delta = LpVariable("dis_gamma_delta",
    lowBound=0, upBound=360)
    dis_delta_epsilon = LpVariable("dis_delta_epsilon",
    lowBound=0, upBound=360)
    dis_beta_delta_ = LpVariable("dis_beta_delta_",
    lowBound=0, upBound=360)
    dis_beta_epsilon_ = LpVariable("dis_beta_epsilon_",
    lowBound=0, upBound=360)
    dis_gamma_epsilon_ = LpVariable("dis_gamma_epsilon_",
    lowBound=0, upBound=360)

    dis_beta_gamma_real = LpVariable("dis_beta_gamma_real",
    lowBound=0, upBound=180)
    dis_gamma_delta_real = LpVariable("dis_gamma_delta_real", lowBound=0,
    upBound=180)
    dis_delta_epsilon_real = LpVariable("dis_delta_epsilon_real", lowBound=0,
    upBound=180)
    dis_beta_delta__real = LpVariable("dis_beta_delta__real", lowBound=0,
    upBound=180)
```

```
dis_beta_epsilon__real = LpVariable("dis_beta_epsilon__real", lowBound=0,
upBound=180)
dis_gamma_epsilon__real = LpVariable("dis_gamma_epsilon__real", lowBound=0,
upBound=180)

w = [None] + [LpVariable(f"w{i}", cat="Binary") for i in range(1, 20)]
v = [None] + [LpVariable(f"v{i}", cat="Binary") for i in range(1, 20)]
y = [None] + [LpVariable(f"y{i}", cat="Binary") for i in range(1, 20)]

z = LpVariable("z", lowBound=0)

# ------------------------------------------------
# 3. Defining the opposite points on the circle
# ------------------------------------------------
    if i < 8:
    model += beta >= 0
    beta_ = beta - 180
else:
    model += beta <= 0
    beta_ = beta + 180

if i // 4 == 0 or i // 4 == 2:
    model += gamma >= 0
    gamma_ = gamma - 180
else:
    model += gamma <= 0
    gamma_ = gamma + 180

if (i // 2) % 2 == 0:
    model += delta >= 0
    delta_ = delta - 180
else:
    model += delta <= 0
    delta_ = delta + 180

if i % 2 == 0:
    model += epsilon >= 0
    epsilon_ = epsilon - 180
else:
    model += epsilon <= 0
    epsilon_ = epsilon + 180

# ------------------------------------------------
# 4. Restrictions
# ------------------------------------------------
M = 400  # big-M constant

# beta & gamma
model += dis_beta_gamma >= beta - gamma
model += dis_beta_gamma >= gamma - beta
model += gamma - beta - dis_beta_gamma >= -M * w[1]
model += beta - gamma - dis_beta_gamma >= -M * (1 - w[1])
model += dis_beta_gamma_real <= dis_beta_gamma
model += dis_beta_gamma_real <= 360 - dis_beta_gamma
model += dis_beta_gamma_real - dis_beta_gamma >= -M * v[1]
model += dis_beta_gamma_real - (360 - dis_beta_gamma) >= -M * (1 - v[1])
```

```
model += dis_beta_gamma_real >= 60
model += dis_beta_gamma_real <= z

# gamma & delta
model += dis_gamma_delta >= gamma - delta
model += dis_gamma_delta >= delta - gamma
model += delta - gamma - dis_gamma_delta >= -M * w[2]
model += gamma - delta - dis_gamma_delta >= -M * (1 - w[2])
model += dis_gamma_delta_real <= dis_gamma_delta
model += dis_gamma_delta_real <= 360 - dis_gamma_delta
model += dis_gamma_delta_real - dis_gamma_delta >= -M * v[2]
model += dis_gamma_delta_real - (360 - dis_gamma_delta) >= -M * (1 - v[2])
model += dis_gamma_delta_real >= 60
model += dis_gamma_delta_real <= z

# delta & epsilon
model += dis_delta_epsilon >= delta - epsilon
model += dis_delta_epsilon >= epsilon - delta
model += epsilon - delta - dis_delta_epsilon >= -M * w[3]
model += delta - epsilon - dis_delta_epsilon >= -M * (1 - w[3])
model += dis_delta_epsilon_real <= dis_delta_epsilon
model += dis_delta_epsilon_real <= 360 - dis_delta_epsilon
model += dis_delta_epsilon_real - dis_delta_epsilon >= -M * v[3]
model += dis_delta_epsilon_real - (360 - dis_delta_epsilon) >= -M * (1 - v[3])
model += dis_delta_epsilon_real >= 60
model += dis_delta_epsilon_real <= z

# beta & delta'
model += dis_beta_delta_ >= beta - delta_
model += dis_beta_delta_ >= delta_ - beta
model += delta_ - beta - dis_beta_delta_ >= -M * w[4]
model += beta - delta_ - dis_beta_delta_ >= -M * (1 - w[4])
model += dis_beta_delta__real <= dis_beta_delta_
model += dis_beta_delta__real <= 360 - dis_beta_delta_
model += dis_beta_delta__real - dis_beta_delta_ >= -M * v[4]
model += dis_beta_delta__real - (360 - dis_beta_delta_) >= -M * (1 - v[4])
model += dis_beta_delta__real >= 60
model += dis_beta_delta__real <= z

# beta & epsilon'
model += dis_beta_epsilon_ >= beta - epsilon_
model += dis_beta_epsilon_ >= epsilon_ - beta
model += epsilon_ - beta - dis_beta_epsilon_ >= -M * w[5]
model += beta - epsilon_ - dis_beta_epsilon_ >= -M * (1 - w[5])
model += dis_beta_epsilon__real <= dis_beta_epsilon_
model += dis_beta_epsilon__real <= 360 - dis_beta_epsilon_
model += dis_beta_epsilon__real - dis_beta_epsilon_ >= -M * v[5]
model += dis_beta_epsilon__real - (360 - dis_beta_epsilon_) >= -M * (1 - v[5])
model += dis_beta_epsilon__real >= 60
model += dis_beta_epsilon__real <= z

# gamma & epsilon'
model += dis_gamma_epsilon_ >= gamma - epsilon_
model += dis_gamma_epsilon_ >= epsilon_ - gamma
model += epsilon_ - gamma - dis_gamma_epsilon_ >= -M * w[6]
model += gamma - epsilon_ - dis_gamma_epsilon_ >= -M * (1 - w[6])
```

```
    model += dis_gamma_epsilon__real <= dis_gamma_epsilon_
    model += dis_gamma_epsilon__real <= 360 - dis_gamma_epsilon_
    model += dis_gamma_epsilon__real - dis_gamma_epsilon_ >= -M * v[6]
    model += dis_gamma_epsilon__real - (360 - dis_gamma_epsilon_) >= -M * (1 - v[6])
    model += dis_gamma_epsilon__real >= 60
    model += dis_gamma_epsilon__real <= z

    # Other
    model += epsilon - 60 >= -M * y[1]
    model += -epsilon - 60 >= -M * (1 - y[1])
    model += beta - 60 >= -M * y[2]
    model += -beta - 60 >= -M * (1 - y[2])

    model += z >= beta
    model += z >= -beta
    model += z >= epsilon
    model += z >= -epsilon
    model += z >= gamma_
    model += z >= -gamma_
    model += z >= delta_
    model += z >= -delta_

    # ------------------------------------------------
    # 4. Objective
    # ------------------------------------------------
    model += z

    # ------------------------------------------------
    # 5. Solve
    # ------------------------------------------------
    model.solve(PULP_CBC_CMD(msg=False))

    results.append({
        "i": i,
        "Status": LpStatus[model.status],
        "z": value(model.objective),
        "beta": value(beta),
        "gamma": value(gamma),
        "delta": value(delta),
        "epsilon": value(epsilon)
    })

# ------------------------------------------------
# 6. Summarize results
# ------------------------------------------------
df = pd.DataFrame(results)
print(df)

best_row = df.loc[df["z"].idxmin()]
print("\nBest solution found for i =", int(best_row["i"]))
print(best_row)
```

A.3. Code for quadratic optimization of two-dimensional flows

The following code is used in Section 6.3 as a general model to find two-dimensional nowhere-zero flows on certain graphs.

```
import gurobipy as gp
from gurobipy import GRB
import numpy
import matplotlib.pyplot as plt

# Create a new model
m = gp.Model("Flow")
edges = 30  # Number of edges

x = [m.addVar(lb=-2, ub=2, name=f"x_{i}") for i in range(edges)]
y = [m.addVar(lb=-2, ub=2, name=f"y_{i}") for i in range(edges)]
t = m.addVar(lb=0, ub=2, name="t")
t.start=2.3

m.Params.NonConvex = 2
m.Params.TimeLimit = 90*60  # 90 minutes

# Set objective:
obj = t
m.setObjective(obj, GRB.MINIMIZE)

# Constraints for all graphs using quadratic constraints
for i in range(edges):
    m.addQConstr(x[i]**2 + y[i]**2 <= t**2, f"t_{i}")
    m.addQConstr(x[i]**2 + y[i]**2 >= 1, f"1_{i}")

# Graph constraints: uncomment the constraints for chosen graph


# -----WAGNER GRAPH-----
# m.addConstr(x[0]-x[1]-x[7] == 0, "x_v1")
# m.addConstr(x[1]-x[2]-x[6] == 0, "x_v2")
# m.addConstr(x[2]-x[3]-x[5] == 0, "x_v3")
# m.addConstr(x[3]-x[0]-x[4] == 0, "x_v4")
# m.addConstr(x[7]+x[11]-x[8] == 0, "x_v5")
# m.addConstr(x[6]+x[8]-x[9] == 0, "x_v6")
# m.addConstr(x[5]+x[10]-x[11] == 0, "x_v7")
# m.addConstr(x[4]+x[9]-x[10] == 0, "x_v8")

# m.addConstr(y[0]-y[1]-y[7] == 0, "y_v1")
# m.addConstr(y[1]-y[2]-y[6] == 0, "y_v2")
# m.addConstr(y[2]-y[3]-y[5] == 0, "y_v3")
# m.addConstr(y[3]-y[0]-y[4] == 0, "y_v4")
# m.addConstr(y[7]+y[11]-y[8] == 0, "y_v5")
# m.addConstr(y[6]+y[8]-y[9] == 0, "y_v6")
# m.addConstr(y[5]+y[10]-y[11] == 0, "y_v7")
# m.addConstr(y[4]+y[9]-y[10] == 0, "y_v8")

# m.addConstr(t<=1.285)


# Graph constraints: uncomment for one graph
```

```
# -----PETERSEN GRAPH-----
# m.addConstr(x[4]-x[0]-x[5] == 0, "x_v1")
# m.addConstr(x[0]-x[1]-x[6] == 0, "x_v2")
# m.addConstr(x[1]-x[2]-x[7] == 0, "x_v3")
# m.addConstr(x[2]-x[3]-x[8] == 0, "x_v4")
# m.addConstr(x[3]-x[4]-x[9] == 0, "x_v5")
# m.addConstr(x[5]+x[14]-x[10] == 0, "x_v6")
# m.addConstr(x[6]+x[11]-x[12] == 0, "x_v7")
# m.addConstr(x[7]+x[13]-x[14] == 0, "x_v8")
# m.addConstr(x[8]+x[10]-x[11] == 0, "x_v9")
# m.addConstr(x[9]+x[12]-x[13] == 0, "x_v10")

# m.addConstr(y[4]-y[0]-y[5] == 0, "y_v1")
# m.addConstr(y[0]-y[1]-y[6] == 0, "y_v2")
# m.addConstr(y[1]-y[2]-y[7] == 0, "y_v3")
# m.addConstr(y[2]-y[3]-y[8] == 0, "y_v4")
# m.addConstr(y[3]-y[4]-y[9] == 0, "y_v5")
# m.addConstr(y[5]+y[14]-y[10] == 0, "y_v6")
# m.addConstr(y[6]+y[11]-y[12] == 0, "y_v7")
# m.addConstr(y[7]+y[13]-y[14] == 0, "y_v8")
# m.addConstr(y[8]+y[10]-y[11] == 0, "y_v9")
# m.addConstr(y[9]+y[12]-y[13] == 0, "y_v10")


# -----ISAACS SNARK / FLOWER SNARK J_5-----
# m.addConstr(x[14] - x[0] - x[15] == 0, "x_v1")
# m.addConstr(x[0] + x[23] - x[1] == 0, "x_v2")
# m.addConstr(x[1] - x[2] - x[21] == 0, "x_v3")
# m.addConstr(x[2] - x[3] - x[16] == 0, "x_v4")
# m.addConstr(x[3] + x[20] - x[4] == 0, "x_v5")
# m.addConstr(x[4] - x[5] - x[29] == 0, "x_v6")
# m.addConstr(x[5] - x[6] - x[17] == 0, "x_v7")
# m.addConstr(x[6] + x[21] - x[7] == 0, "x_v8")
# m.addConstr(x[7] - x[8] - x[22] == 0, "x_v9")
# m.addConstr(x[8] - x[9] - x[18] == 0, "x_v10")
# m.addConstr(x[9] + x[29] - x[10] == 0, "x_v11")
# m.addConstr(x[10] - x[11] - x[23] == 0, "x_v12")
# m.addConstr(x[11] - x[12] - x[19] == 0, "x_v13")
# m.addConstr(x[12] + x[22] - x[13] == 0, "x_v14")
# m.addConstr(x[13] - x[14] - x[20] == 0, "x_v15")
# m.addConstr(x[15] + x[28] - x[24] == 0, "x_v16")
# m.addConstr(x[16] + x[24] - x[25] == 0, "x_v17")
# m.addConstr(x[17] + x[25] - x[26] == 0, "x_v18")
# m.addConstr(x[18] + x[26] - x[27] == 0, "x_v19")
# m.addConstr(x[19] + x[27] - x[28] == 0, "x_v20")

# m.addConstr(y[14] - y[0] - y[15] == 0, "y_v1")
# m.addConstr(y[0] + y[23] - y[1] == 0, "y_v2")
# m.addConstr(y[1] - y[2] - y[21] == 0, "y_v3")
# m.addConstr(y[2] - y[3] - y[16] == 0, "y_v4")
# m.addConstr(y[3] + y[20] - y[4] == 0, "y_v5")
# m.addConstr(y[4] - y[5] - y[29] == 0, "y_v6")
# m.addConstr(y[5] - y[6] - y[17] == 0, "y_v7")
# m.addConstr(y[6] + y[21] - y[7] == 0, "y_v8")
# m.addConstr(y[7] - y[8] - y[22] == 0, "y_v9")
# m.addConstr(y[8] - y[9] - y[18] == 0, "y_v10")
```

```
# m.addConstr(y[9] + y[29] - y[10] == 0, "y_v11")
# m.addConstr(y[10] - y[11] - y[23] == 0, "y_v12")
# m.addConstr(y[11] - y[12] - y[19] == 0, "y_v13")
# m.addConstr(y[12] + y[22] - y[13] == 0, "y_v14")
# m.addConstr(y[13] - y[14] - y[20] == 0, "y_v15")
# m.addConstr(y[15] + y[28] - y[24] == 0, "y_v16")
# m.addConstr(y[16] + y[24] - y[25] == 0, "y_v17")
# m.addConstr(y[17] + y[25] - y[26] == 0, "y_v18")
# m.addConstr(y[18] + y[26] - y[27] == 0, "y_v19")
# m.addConstr(y[19] + y[27] - y[28] == 0, "y_v20")


# ------TIETZE'S GRAPH------
m.addConstr(x[0] + x[7] - x[1] == 0, "x_v1")
m.addConstr(x[1] - x[2] - x[8] == 0, "x_v2")
m.addConstr(x[2] - x[0] - x[3] == 0, "x_v3")
m.addConstr(x[3] - x[4] - x[12] == 0, "x_v4")
m.addConstr(x[4] - x[5] - x[11] == 0, "x_v5")
m.addConstr(x[5] - x[6] - x[10] == 0, "x_v6")
m.addConstr(x[6] - x[7] - x[9] == 0, "x_v7")
m.addConstr(x[8] + x[15] - x[16] == 0, "x_v8")
m.addConstr(x[9] + x[13] - x[14] == 0, "x_v9")
m.addConstr(x[10] + x[16] - x[17] == 0, "x_v10")
m.addConstr(x[11] + x[14] - x[15] == 0, "x_v11")
m.addConstr(x[12] + x[17] - x[13] == 0, "x_v12")

m.addConstr(y[0] + y[7] - y[1] == 0, "y_v1")
m.addConstr(y[1] - y[2] - y[8] == 0, "y_v2")
m.addConstr(y[2] - y[0] - y[3] == 0, "y_v3")
m.addConstr(y[3] - y[4] - y[12] == 0, "y_v4")
m.addConstr(y[4] - y[5] - y[11] == 0, "y_v5")
m.addConstr(y[5] - y[6] - y[10] == 0, "y_v6")
m.addConstr(y[6] - y[7] - y[9] == 0, "y_v7")
m.addConstr(y[8] + y[15] - y[16] == 0, "y_v8")
m.addConstr(y[9] + y[13] - y[14] == 0, "y_v9")
m.addConstr(y[10] + y[16] - y[17] == 0, "y_v10")
m.addConstr(y[11] + y[14] - y[15] == 0, "y_v11")
m.addConstr(y[12] + y[17] - y[13] == 0, "y_v12")


# -----ANOTHER SNARK-----
# m.addConstr(x[4] - x[1] - x[5] == 0, "x_v1")
# m.addConstr(x[1] - x[2] - x[8] == 0, "x_v2")
# m.addConstr(x[3] + x[13] - x[4] == 0, "x_v3")
# m.addConstr(x[2] + x[6] - x[3] == 0, "x_v4")
# m.addConstr(x[5] - x[6] - x[7] == 0, "x_v5")
# m.addConstr(x[12] + x[19] - x[13] == 0, "x_v6")
# m.addConstr(x[7] + x[15] + x[20] == 0, "x_v7")
# m.addConstr(x[8] - x[9] - x[14] == 0, "x_v8")
# m.addConstr(x[8] - x[9] - x[14] == 0, "x_v9")
# m.addConstr(x[0] + x[9] - x[10] == 0, "x_v10")
# m.addConstr(x[17] - x[18] - x[20] == 0, "x_v11")
# m.addConstr(x[14] + x[16] - x[17] == 0, "x_v12")
# m.addConstr(x[11] - x[12] - x[16] == 0, "x_v13")
# m.addConstr(x[10] - x[11] - x[15] == 0, "x_v14")
```

```
# m.addConstr(y[4] - y[1] - y[5] == 0, "y_v1")
# m.addConstr(y[1] - y[2] - y[8] == 0, "y_v2")
# m.addConstr(y[3] + y[13] - y[4] == 0, "y_v3")
# m.addConstr(y[2] + y[6] - y[3] == 0, "y_v4")
# m.addConstr(y[5] - y[6] - y[7] == 0, "y_v5")
# m.addConstr(y[12] + y[19] - y[13] == 0, "y_v6")
# m.addConstr(y[7] + y[15] + y[20] == 0, "y_v7")
# m.addConstr(y[8] - y[9] - y[14] == 0, "y_v8")
# m.addConstr(y[8] - y[9] - y[14] == 0, "y_v9")
# m.addConstr(y[0] + y[9] - y[10] == 0, "y_v10")
# m.addConstr(y[17] - y[18] - y[20] == 0, "y_v11")
# m.addConstr(y[14] + y[16] - y[17] == 0, "y_v12")
# m.addConstr(y[11] - y[12] - y[16] == 0, "y_v13")
# m.addConstr(y[10] - y[11] - y[15] == 0, "y_v14")


# -----BIDIAKIS CUBE-----
# m.addConstr(x[0] - x[1] - x[6] == 0, "x_v1")
# m.addConstr(x[1] - x[2] - x[7] == 0, "x_v2")
# m.addConstr(x[2] - x[3] - x[8] == 0, "x_v3")
# m.addConstr(x[3] - x[4] - x[9] == 0, "x_v4")
# m.addConstr(x[4] - x[0] - x[5] == 0, "x_v5")
# m.addConstr(x[5] - x[10] - x[13] == 0, "x_v6")
# m.addConstr(x[6] + x[10] - x[11] == 0, "x_v7")
# m.addConstr(x[11] - x[12] - x[14] == 0, "x_v8")
# m.addConstr(x[7] + x[14] - x[15] == 0, "x_v9")
# m.addConstr(x[8] + x[15] - x[16] == 0, "x_v10")
# m.addConstr(x[9] + x[16] - x[17] == 0, "x_v11")
# m.addConstr(x[12] + x[13] + x[17] == 0, "x_v12")

# m.addConstr(y[0] - y[1] - y[6] == 0, "y_v1")
# m.addConstr(y[1] - y[2] - y[7] == 0, "y_v2")
# m.addConstr(y[2] - y[3] - y[8] == 0, "y_v3")
# m.addConstr(y[3] - y[4] - y[9] == 0, "y_v4")
# m.addConstr(y[4] - y[0] - y[5] == 0, "y_v5")
# m.addConstr(y[5] - y[10] - y[13] == 0, "y_v6")
# m.addConstr(y[6] + y[10] - y[11] == 0, "y_v7")
# m.addConstr(y[11] - y[12] - y[14] == 0, "y_v8")
# m.addConstr(y[7] + y[14] - y[15] == 0, "y_v9")
# m.addConstr(y[8] + y[15] - y[16] == 0, "y_v10")
# m.addConstr(y[9] + y[16] - y[17] == 0, "y_v11")
# m.addConstr(y[12] + y[13] + y[17] == 0, "y_v12")

# Optimization
m.optimize()

if m.Status == GRB.OPTIMAL:
    for v in m.getVars():
        print(f"{v.VarName} {v.X:g}")
    print(f"Obj: {m.ObjVal:g}")
else:
    print(f"Model not solved to optimality. Status: {m.Status}")

for v in m.getVars():
    print(f"{v.VarName} {v.X:g}")
```

```
print(f"Obj: {m.ObjVal:g}")

-----PLOTTING THE OPTIMAL SOLUTION-----
# Get optimal values
x_vals = [v.X for v in x]
y_vals = [v.X for v in y]
t_opt = t.X

# Make plot
plt.figure(figsize=(10, 10))

# Plot circles
circle1 = plt.Circle((0, 0), 1, color='blue', fill=False, linestyle='--',
label='Circle radius 1')
plt.gca().add_patch(circle1)
circle_t = plt.Circle((0, 0), t_opt, color='red', fill=False, linestyle='-',
label=f'Circle radius t={t_opt:.2f}')
plt.gca().add_patch(circle_t)

# Plot points (x_i, y_i)
for i in range(edges):
    plt.scatter(x_vals[i], y_vals[i], color='green', zorder=5)
    plt.text(x_vals[i], y_vals[i], f' ({x_vals[i]:.2f},
    {y_vals[i]:.2f})', fontsize=8, ha='right')

# Finish plot
plt.xlim(-2.5, 2.5)
plt.ylim(-2.5, 2.5)
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)
plt.grid(color='gray', linestyle='--', linewidth=0.5)
plt.title('Optimal solution: ...') # name of graph
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.axis('equal')
plt.show()
```